

# XVT-Design++<sup>TM</sup>

**AN INTERACTIVE C++ APPLICATION FRAMEWORK FROM THE #1 COMPANY IN MULTI-PLATFORM DEVELOPMENT TOOLS.**

# XVT

# XVT-Design++

Volume

# 1

# USING XVT-DESIGN++

---

XVT - THE PORTABLE GUI DEVELOPMENT SOLUTION

---



# Interactive Design Tool

# USING XVT-DESIGN++

---

XVT - THE PORTABLE GUI DEVELOPMENT SOLUTION

---



## Copyrights

© 1992 - 1993 XVT Software Inc. All rights reserved.

The XVT-Design++ application program interface, XVT manuals and technical literature may not be reproduced in any form or by any means except by permission in writing from XVT Software Inc.

XVT++ and XVT-Design++ are trademarks of XVT Software Inc. Other product names mentioned in this document are trademarks or registered trademarks of their respective holders.

## Published By

XVT Software Inc.  
Box 18750  
Boulder, CO 80308  
(303) 443-4223  
(303) 443-0969 (FAX)

## Credits

### Original Concept and Design:

Marc Rochkind

### Architecture and Specification:

Mike Ernst, Jack Unrue, Scott Meyer, Martin Brunecky

### Engineers:

Jack Unrue, Doug Young, John Humbrecht, Steve Archuleta

### Documentation:


Lynn Merrill, Jim Ruffing, Dave Welsch

### Product Team:

Dave Locke, Mike Ernst, Connie Leserman, Eva Johnson,  
Peggy Reed, Lissa Sinnickson, Jonathan Auerbach,  
Catherine Connor

## Printing History

First Printing ..... June, 1993 ..... XVT-Design++ Version 1.0

This manual is printed on recycled paper by  
Continental Graphics, Broomfield, Colorado. 

# ***XVT-DESIGN++***

## **CONTENTS**

<b>Chapter 1. Introduction.....</b>	<b>1</b>
1.1. What is XVT-Design++? .....	1
1.2. Using This Manual .....	2
1.2.1. Conventions Used in This Manual .....	3
1.2.2. On-line Help .....	3
1.3. Key Features in Version 1.0.....	4
1.3.1. Resource Definition and Layout .....	4
1.3.2. Integrated Code Editing .....	4
1.3.3. Connections.....	4
1.3.4. TestMode.....	4
1.3.5. Complete Makefile Generation .....	4
1.3.6. Native Control Rendering .....	5
1.4. Obtaining Help .....	5
<b>Chapter 2. XVT-Design++ Concepts .....</b>	<b>7</b>
2.1. C++ and XVT++ Overview .....	7
2.1.1. Classes .....	7
2.1.2. Subclasses.....	9
2.1.3. Event Handler Member Functions .....	10
2.1.4. Learning More About C++.....	12
2.2. Interface Object Attributes .....	13
2.2.1. Geometry .....	13
2.2.2. Title .....	13
2.2.3. Resource Identifier .....	13
2.2.4. Other Attributes.....	13
2.3. User Interface Code.....	14
2.3.1. Integrated Code Editing .....	14
2.3.2. Generated Code .....	14

2.3.3.	Tags.....	14
2.3.4.	Action Code .....	15
2.3.5.	Context.....	15
2.4.	Test Mode .....	15
2.4.1.	Connections .....	15
<b>Chapter 3. Tutorial.....</b>		<b>17</b>
3.1.	Building XVT-Design++ Applications.....	17
3.2.	The Hello Application .....	18
3.3.	Creating a New Project .....	18
3.4.	Creating a Menubar and Menus.....	19
3.4.1.	The Menubar Editor.....	20
3.4.2.	The Menu Editor.....	20
3.5.	Saving the Project .....	26
3.6.	Creating Containers .....	26
3.6.1.	Creating the Message Window .....	27
3.6.2.	Creating the Other Choices Dialog.....	29
3.6.3.	Creating an About Hello Dialog .....	32
3.7.	Setting Application Attributes .....	33
3.8.	Setting Connections Between Interface Objects.....	34
3.8.1.	Task Menubar Connections .....	35
3.8.2.	Initial Message Window Connection.....	37
3.8.3.	Message Window Menubar .....	37
3.8.4.	Message Window Connections.....	38
3.8.5.	Other Choices Dialog Connections.....	39
3.8.6.	About Hello Dialog Connection .....	40
3.9.	Running TestMode .....	40
3.10.	Attaching Action Code to Tags .....	42
3.10.1.	Storing Information in the Message Window...43	
3.10.2.	Displaying the Message .....	45
3.10.3.	Changing the Message with Menu Commands .....	46
3.10.4.	Changing the Message with the Choices Dialog.....	47
3.10.5.	Adding a #include .....	50
3.11.	Generating the Application.....	50
3.11.1.	Setting the Application Name .....	51
3.11.2.	Generating the Source Files .....	52
3.12.	Building and Running the Application .....	52
<b>Chapter 4. Usage.....</b>		<b>53</b>
4.1.	Project Files .....	53
4.1.1.	Creating New Projects .....	54

4.1.2.	Working with Multiple Projects .....	54
4.2.	The Action Code Editor (ACE) .....	54
4.2.1.	Invoking the ACE .....	54
4.2.2.	ACE Controls .....	55
4.3.	Creating Windows, Dialogs, and Controls .....	60
4.3.1.	Creating Windows and Dialogs .....	60
4.3.2.	Creating Controls .....	60
4.4.	Layout Commands .....	61
4.4.1.	Alignment .....	61
4.4.2.	Spacing .....	63
4.4.3.	Grid .....	63
4.5.	Setting Interface Object Attributes .....	64
4.5.1.	Common Attributes .....	65
4.5.2.	Control Attributes .....	66
4.5.3.	Text Edit Attributes .....	68
4.5.4.	Dialog Box Attributes .....	71
4.5.5.	Window Attributes .....	72
4.6.	Creation Order .....	74
4.7.	The Menu Editor .....	75
4.7.1.	Menubar Editor .....	76
4.7.2.	Menu Editor .....	77
4.7.3.	Menu Attributes .....	80
4.8.	String Resources .....	82
4.8.1.	Strings .....	82
4.8.2.	String Lists .....	83
4.9.	Userdata Strings .....	85
4.9.1.	Creating Userdata .....	86
4.9.2.	Editing Userdata .....	86
4.9.3.	Userdata Labels .....	87
4.9.4.	Generating Code with Userdata .....	89
4.10.	TestMode .....	89
4.10.1.	Entering TestMode .....	89
4.10.2.	Leaving TestMode .....	90
4.10.3.	Special Considerations for TestMode .....	91
4.11.	File Generation .....	91
4.11.1.	Destination Directory .....	91
4.11.2.	Filenames .....	92
4.11.3.	Types of Generated Files .....	93
4.11.4.	Choosing Files to Generate .....	94
4.11.5.	Makefiles .....	95

<b>Chapter 5. Reference.....</b>	<b>99</b>
5.1. Menu Commands .....	99
5.1.1. File Menu .....	99
5.1.2. Edit Menu .....	101
5.1.3. Tools Menu .....	102
5.1.4. Window Menu .....	103
5.1.5. Controls Menu .....	103
5.1.6. Layout Menu .....	104
5.1.7. Font and Style Menus .....	105
5.2. Interface Objects and Tags .....	106
5.2.1. Tag Descriptions .....	106
5.2.2. Interface Object/Tag Pairs .....	109
5.3. Variables and Constants in Action Code .....	111
5.4. The Configuration File.....	111
5.4.1. Name and Location .....	112
5.4.2. Format .....	112
5.4.3. Available Options .....	113
5.4.4. Configuration File Example.....	116
<b>Index .....</b>	<b>117</b>

# 1

---

## INTRODUCTION

This chapter provides a brief overview of XVT-Design++, and suggests how to use this manual most effectively.

### 1.1. What is XVT-Design++?

XVT-Design++ is a graphical, interactive design tool and application generator. It simplifies the design and implementation of graphical-user-interface-based applications in three major ways:

- XVT-Design++ lets you create the user-interface objects of your application (windows, controls, menus, and so on) graphically and interactively, rather than by programming manually.
- XVT-Design++ provides a TestMode that lets you preview your application's user interface without separate compilation and linking steps. You can use XVT-Design++ to build and refine application prototypes rapidly without writing any code.
- XVT-Design++ generates subclasses and other source code for your application's user interface. Instead of rewriting "generic" user-interface code by hand for each new application, you can use XVT-Design++ to create this code automatically.

Without XVT-Design++, it is necessary to write resource language source code by hand to define the necessary GUI resources for an application in a portable manner. Since you cannot visually inspect your resources until you run the application, this is a cumbersome, iterative process.

With XVT-Design++, you create a "project" file containing the GUI resources for your application. For each project, you can create any

number of dialog boxes and windows. Then you draw the needed controls right on your computer screen, placing them just where you want within a window or dialog box—without ever having to calculate the screen coordinates numerically.

XVT-Design++ automatically generates a complete set of source files for your application:

- “Generated” C++ code for the application’s user interface
- C++ header files containing subclasses, member functions, and constant declarations
- An XVT Universal Resource Language (URL) file, containing portable definitions of your application’s resources
- A makefile for compiling and linking your application

**See Also:** To learn more about C++ and XVT++, see section 2.1.

## 1.2. Using This Manual

This manual is intended for programmers who have some experience with building and using GUI applications. It assumes familiarity with basic GUI concepts, such as events and resources, and some familiarity with the XVT Portability Toolkit. Since XVT-Design++ generates XVT++-based programs, use this manual in conjunction with the *XVT++ 2.0 Class Library Reference*.

This manual also assumes some experience with C++ programming. Section 2.1 provides a brief overview of C++ and XVT++ programming. That section also provides a list of books that introduce and discuss C++ programming.

Because XVT-Design++ is an XVT application available on multiple platforms, separate installation instructions are provided with each media distribution. This manual assumes you have already installed XVT-Design++ and the XVT Portability Toolkit on your development system.

Along with this introduction, this manual contains the following chapters:

### **Chapter 2: Concepts**

Introduces some terms and concepts unique to XVT-Design++.

### **Chapter 3: Tutorial**

Guides you through the construction of a small but complete GUI-based application.

**Chapter 4: Usage**

Describes how to use all the features of XVT-Design++.

**Chapter 5: Reference**

Summarizes all the menu commands in XVT-Design++ and describes the format of XVT-Design++ configuration files.

**1.2.1. Conventions Used in This Manual**

To save space and maintain continuity, we have used one platform (Macintosh) for the screen illustrations in this manual. For the same reasons, we have used a consistent set of terms for similar elements that may have platform-specific names. For example, we refer to both a Macintosh “folder” and an MS-Windows “directory” as a directory.

The following typographic conventions indicate different types of information:

---

**code**

This typestyle represents code, including expressions and names of functions, attributes, variables, and structures.

**filenames**

Bold type is used for filenames and directory names.

*emphasis*

Italics are used for emphasis and the names of documents.

- ▼ This triangle symbol marks the beginning of a procedure having numbered steps. These symbols can help you quickly locate “how-to” information.

**Note:** An italic heading like this marks a standard kind of information: a Note, Caution, Example, Tip, or See Also (cross-reference).

---

**1.2.2. On-line Help**

In addition to this manual, summary documentation is available for all XVT-Design++ features and menu items in an on-line Help utility.

## 1.3. Key Features in Version 1.0

### 1.3.1. Resource Definition and Layout

XVT-Design++ 1.0 allows the application programmer to graphically and interactively create user-interface objects. These interface objects include windows, dialogs, controls, and menus. The XVT++ 2.0 Class Library provides class definitions and default event handler member functions for these and other interface objects. You can achieve substantial savings of time and effort with XVT-Design++, compared to manual programming.

### 1.3.2. Integrated Code Editing

XVT-Design++ 1.0 includes an Action Code Editor for creating and editing your application's user interface source code. With the Action Code Editor, subclasses and event handler member functions can be easily and quickly defined and refined. Since source code is integrated with resources in the project file, regenerating your application's code will not overwrite your existing code. XVT-Design++ can be used throughout the entire development of your application, from prototyping to cross-platform porting and polishing.

### 1.3.3. Connections

Version 1.0 provides *connections* between GUI objects. A connection creates or removes a window or dialog when some event occurs. For example, a connection for a push button could present a dialog when the button is pressed. After you define connections, XVT-Design++ automatically generates source code for them.

### 1.3.4. TestMode

XVT-Design++ 1.0 includes a testing mode that lets you check the appearance and behavior of your application's user interface resources—without compiling and linking. In TestMode, XVT-Design++ simulates all connections you have defined, and presents your application's windows, dialogs, and menus as they will appear at execution time.

### 1.3.5. Complete Makefile Generation

XVT-Design++ generates working makefiles that are ready for compilation on all supported platforms. Makefiles are generated

automatically from predefined templates. User-defined templates can also be used to modify the generated makefiles to meet specific compilation needs.

### **1.3.6. Native Control Rendering**

In layout windows and TestMode, all XVT API controls are rendered in their native appearance. Controls have the same appearance in XVT-Design++ 1.0 as they will when your application is compiled and executed.

## **1.4. Obtaining Help**

If you encounter a problem using XVT-Design++, you can contact XVT Software Customer Support in several different ways:

- Telephone us at (303) 443-8988 (8 AM to 5 PM, MST, Monday-Friday)
- Send us electronic mail via the Internet at [techsup@xvt.com](mailto:techsup@xvt.com)
- Send us electronic mail via CompuServe Mail at 75765,1233
- Send us electronic mail via our Bulletin Board System (BBS) at (303) 443-9083 (2400 baud) or (303) 443-7780 (9600 baud). Note: Before you can use the BBS, you must contact us to get a BBS login.
- Write us at XVT Software Inc., P.O. Box 18750, Boulder, CO 80308

When you contact us, be ready to supply the following information:

- Your software serial number (found on your distribution media)
- Your platform type
- The software version you are running
- Any relevant information regarding symptoms, including the number displayed with internal error messages such as this: Internal XVT-Design++ Error: 37007-408323

Please note that only one individual per purchased copy of XVT-Design++ may request support. Questions will be taken only from the individual named on your software registration form.

Feel free to contact XVT Software Customer Support if you would like to request a software enhancement or suggest a change to this document.



# 2

---

## XVT-DESIGN++ CONCEPTS

This chapter introduces some terms and concepts that are used throughout this manual.

### 2.1. C++ and XVT++ Overview

This section introduces C++ programming as implemented within the XVT++ environment. XVT++ provides a complete C++ interface to the functionality offered by the XVT Portability Toolkit. If you are an experienced C++ programmer, you may want to skip ahead to section 2.2.

**Note:** XVT++ is a component of XVT-Design++. The XVT++ 2.0 Class Library has been specifically designed to work with XVT-Design++.

#### 2.1.1. Classes

The *XVT++ 2.0 Class Library Reference* provides a complete discussion of the XVT++ classes, including their structure, usage, constructors, and operators. This document is an invaluable reference when working through the examples in this manual as well as with XVT-Design++ programming in general.

XVT++ uses C++ classes to represent user-interface objects. Windows, menus, controls, strings, and pens are examples of objects/classes. A class incorporates a complete description of an interface object, including the following:

- Characteristics (i.e., state) such as size, location, and color
- Behaviors that specify its reaction to events or situations
- Relations between itself and other classes

Classes are related to one another, in terms of degrees of specialization, to form a hierarchy. General classes are toward the top of the hierarchy and more specific classes toward the bottom. Classes that are more specialized typically contain a more detailed description. Additional characteristics and/or behaviors can be incorporated into the more specialized class. Also, more detailed behaviors can be specified in place of previously defined ones.

The complete XVT++ hierarchy appears on page 4 of the *XVT++ 2.0 Class Library Reference*. A portion of that hierarchy is shown in Figure 1.

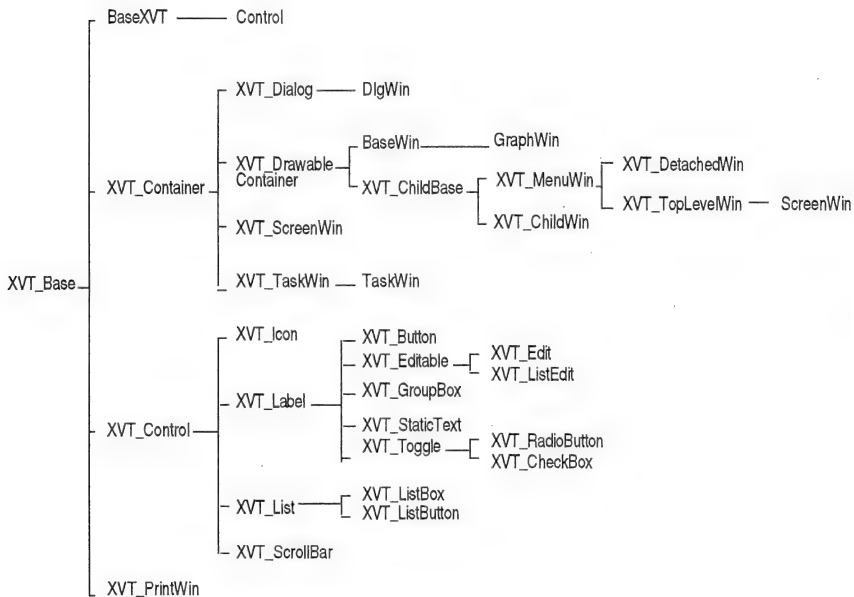


Figure 1. A Portion of the XVT++ Class Hierarchy

In relation to a specific class, a class that is a generalization of it is called a *superclass*. `XVT_Control` is a superclass of `XVT_List`. Classes that are specializations of it are called *subclasses*. `XVT_ListBox` and `XVT_ListButton` are subclasses of `XVT_List`. An XVT++ class has only one superclass but any number of subclasses.

**Note:** Although XVT++ does not use multiple inheritance, you can use it in your applications. Be careful with when and how you use multiple inheritance, however, because XVT++ has been designed to minimize your need for it.

## 2.1.2. Subclasses

Subclasses are specializations of their superclass. For example, XVT++ has an `XVT_Control` class, whose subclasses include `XVT_Icon`, `XVT_Label`, `XVT_List`, and `XVT_Scrollbar`. `XVT_Control` is obviously an interface object that generally describes the characteristics and behaviors of a control. Each of its subclasses is a more specific description of a particular type of control.

The XVT++ hierarchy incorporates classes to describe all of the user-interface objects that make up the XVT API and that will be useful to your application. To develop your GUI, you will extend this hierarchy as required. You can do this one of two ways:

- By creating instances of classes in the XVT++ hierarchy
- By extending the XVT++ hierarchy with even more specialized classes, then creating instances of those

A typical XVT++ application consists of a subclass of `XVT_TaskWin` and one or more subclasses of the GUI container classes: `XVT_ToplevelWin`, `XVT_DetachedWin`, and `XVT_Dialog`. Subclasses of controls will probably be needed as well. Each subclass is specialized by defining behaviors (event handler member functions) for it that are unique to the application. Your event handler member functions replace the general ones supplied by XVT++.

After you have created some subclasses, the extended XVT++ class hierarchy might appear as shown in Figure 2. The user-defined subclasses are prefaced with “My\_”.

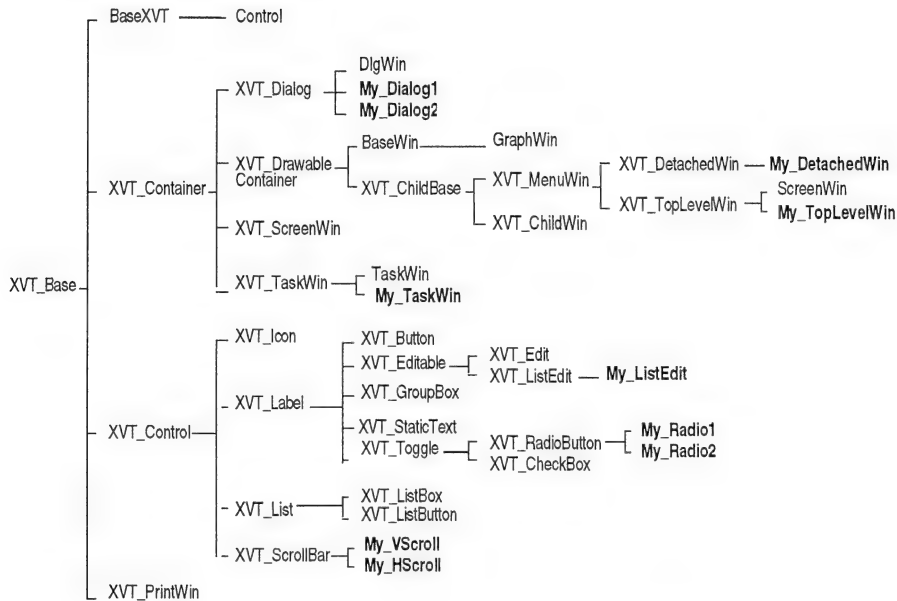


Figure 2. The XVT++ Class Hierarchy with Extensions

### 2.1.3. Event Handler Member Functions

Within each subclass you create, you will write *event handler member functions* to specify and control the behavior of the interface object. Event handler member functions specify the interface object's behavior in response to an event. For instance, what should happen when a user selects an item from a menu?

The classes in the XVT++ hierarchy include default event handler member functions for each event type applicable to the interface object. In most cases, this default behavior is quite limited. Generally, nothing happens when the default event handler member function processes an event.

When you want objects to react to an event in a certain way, it is up to you to write event handler member functions for them. Once you have used XVT-Design++ to lay out your resources, most of your effort will be associated with writing application-specific event handler member functions.

The *XVT++ 2.0 Class Library Reference* lists the default event handler member functions for each class. Use this as a guide for

what functions you need to write, what parameters the functions should accept, and what each function's return value should be.

### 2.1.3.1. Implementing Application Behavior

XVT++ 2.0 preserves the style of programming used in version 1.1: you are expected to create subclasses that override virtual event handler member functions (the `e_*` functions) to implement whatever behavior your application needs. For many applications, this scheme is satisfactory; however, there are times when other techniques might be preferred.

One such situation is the case where an application has many controls or windows that are very similar, for example 50 text entry fields that collect data for a database query. Creating 50 distinct classes results in much duplicate code. A better solution is to create a single class that changes its behavior based on parameters provided in the constructor, in additional member functions, or in resource user data. You can then create 50 instances of this class, one for each control.

In other cases, the fact that all of an object's behavior must be specified in a single subclass definition causes difficulties. A symptom of this sort of problem would be subclass member functions that all start with `if` or `switch` statements, which cause the member function to behave in completely different ways based on the state of the object.

A cleaner approach is to create what are known as behavior or delegate objects. A behavior object implements a single type of behavior; it has no `switch` statements. The behavior of an object is changed by replacing behavior objects instead of taking different paths through switch logic.

**Tip:** The simplest way to implement behaviors is to create an abstract behavior class that has member functions corresponding to the event handling member functions present in the XVT++ object. Each actual behavior will be a subclass of this behavior class.

The subclass of the XVT++ class is very simple. It adds storage for a current behavior pointer and implementations of the event handling member functions that just call the corresponding function in the current behavior. When the subclass is instantiated, it installs the behavior corresponding to the start state. As the object is manipulated, the current behavior is called and can manipulate the XVT++ object subclass as required, including switching the current behavior.

### 2.1.3.2. GUI Objects and Container Classes

With the exception of menus, all GUI objects have at least `e_create` and `e_destroy` event handler member functions, which are called when the interface object is created and just before it is destroyed, respectively. All GUI objects, including menus, also have constructor and destructor member functions. These are called when the C++ object is constructed and destructed.

Any of the GUI container classes can contain controls and, in the case of windows, text edit objects and child windows. Like GUI containers, controls have their own event handler member functions that are overridden by user subclasses. Most controls have at least `e_create`, `e_destroy`, and `e_action` event handler member functions.

All GUI objects—controls, windows, and dialogs—have a two-phase creation protocol. The two-phase protocol prevents a problem that can occur when the window system causes recursion in a C++ constructor: callbacks from the window system can cause the application program to try to use an object that is not yet completely constructed. In the two-phase protocol, the GUI object is first created with the C++ `new` operator, then initialized with the `Init` member function. The object's `e_create` member function is called before `Init` returns.

### 2.1.4. Learning More About C++

If you need help with C++ programming, we recommend the following books:

- *C++ Primer* by Lippman, published by Addison-Wesley Co. in 1989.
- *The C++ Programming Language* by Bjarne Stroustrup, published by Addison-Wesley Co. in 1992.
- *C++ Programming Style* by Tom Cargill, published by Addison-Wesley Co. in 1992.
- *Mastering C++* by Horstmann, published by John Wiley & Sons in 1990.
- *Programming in C++* by Dewhurst and Stark, published by Prentice-Hall in 1989.
- *Teaching Yourself C++* by Herbert Schildt, published by Osborne/McGraw-Hill Inc. in 1992.

## 2.2. Interface Object Attributes

All the GUI objects provided by the XVT Portability Toolkit—windows, dialogs, controls, and so on—have a number of attributes that describe their appearance and behavior. These attributes are described in detail in the *XVT Guide*. XVT-Design++ presents dialog boxes with controls for these attributes, allowing you to set the values of the attributes interactively, rather than specifying them programmatically.

### 2.2.1. Geometry

The size and position of windows, dialogs, and controls are specified by rectangles. Size is indicated by the height and width of the rectangle. Position is specified by the horizontal and vertical coordinates of the upper-left corner of the rectangle, with respect to the interface object's container. (Windows and dialogs are contained by the Task Window.)

Instead of describing the size and position of interface objects in resource-description text files, XVT-Design++ lets you create and modify interface objects graphically, like a drawing application.

### 2.2.2. Title

All interface objects have a title string, which is the name of the interface object from the application user's point of view. The interface object may or may not have a visible title, depending on the type and conventions of the native windowing system. (For instance, document windows and buttons almost always have visible titles, but scrollbars do not.)

### 2.2.3. Resource Identifier

Interface objects (i.e., resources) have a second string that is used by the application developer, rather than the user. This string lets you refer to the interface object with a symbolic name, rather than its resource ID number. We refer to this string as the `#define` string. The `#define` string associates a symbolic name with the ID number that is used in the resource file.

### 2.2.4. Other Attributes

Each type of interface object has additional attributes, specific to its type. For instance, windows have attributes that describe their

border decorations, and menus have attributes that specify accelerator and mnemonic keys.

## 2.3. User Interface Code

In addition to creating resources graphically, XVT-Design++ helps you create the program code for your application's user interface objects.

### 2.3.1. Integrated Code Editing

The *Action Code Editor* (ACE) lets you create and edit your user interface source code without leaving XVT-Design++. This ensures that the code you enter is preserved as part of the interface definition.

### 2.3.2. Generated Code

XVT-Design++ creates event handler member functions and other code that provide a framework for your application's user interface. Rather than writing your subclasses and member functions from scratch, you can use XVT-Design++ to generate this code for you.

While using XVT-Design++, you do not actually see the generated code. The code you create is integrated with the generated code when XVT-Design++ produces the source code files for your application.

### 2.3.3. Tags

All user interface objects have a number of associated events, that is, occurrences that the application responds to. XVT-Design++ assigns *event tags* to these events. The event tags correspond to the `e_*` member functions defined in the *XVT++ 2.0 Class Library Reference*.

Each interface object may also have a number of special tags. Special tags do not correspond to any runtime events, but are "markers" in the generated code for an interface object. They indicate positions in the generated code where you may want to insert your own code. In general, special tags apply to other than `e_*` member functions or to the subclass definition. For example, the `Class Decl` special tag marks a location appropriate for adding state variables to the generated subclass.

### 2.3.4. Action Code

Action code is C++ source code that implements some action in response to an event. Action code is added to your application with the Action Code Editor, and incorporated into code generated by XVT-Design++ when your application's source code files are produced.

### 2.3.5. Context

Action code is always associated with a specific context. A context is composed of three parts:

#### Module

A module is a user-interface component that contains other components. Windows, dialogs, menubars, and the application itself are all modules.

#### Object

An object is one of the components contained by a module. Controls and menu items are interface objects. In XVT-Design++, modules are also considered to be objects (in a sense, they contain themselves).

#### Tag

Every object has one or more tags, as described above.

The unique combination of a module, an object, and a tag make up a context. Three list buttons in the Action Code Editor specify the context for action code.

## 2.4. Test Mode

TestMode in XVT-Design++ lets you test your application's user interface without compiling and linking source code. You can rapidly refine the appearance of user interface objects without leaving XVT-Design++.

### 2.4.1. Connections

Rather than interpreting or (executing) your application's source code, TestMode in XVT-Design++ uses *connections* to define relationships between interface objects. Like action code, connections are associated with tags. A connection opens or closes a container (window or dialog) when a tag's event occurs. Connections can also invoke predefined dialogs in XVT++, such as the standard open-file and save-file dialogs.



# 3

---

## TUTORIAL

This tutorial chapter demonstrates how to use XVT-Design++ to build a sample application. You should do the tutorial from beginning to end, preferably in one sitting.

Before doing the tutorial, read Chapter 2 of this manual. That “Concepts” chapter contains definitions of key terms used throughout the tutorial.

Begin the tutorial by running XVT-Design++. If you have not yet installed XVT-Design++, refer to the installation sheet in your XVT-Design++ package for instructions.

### 3.1. Building XVT-Design++ Applications

In general, when you develop an XVT-Design++ application you will do the following:

- Define any custom classes that will serve as additions to the XVT++ class hierarchy (this is done outside of XVT-Design++ using the native editor)
- In XVT-Design++, create a new project file for the application
- Set the project attributes
- Create the menubar(s) and menus
- Create the windows and dialogs
- Create and lay out any controls in the windows and dialogs
- Attach menubars to the windows
- Set connections between interface objects
- Run TestMode to evaluate the project and edit as desired

- Attach action code to event tags (i.e., write application-specific event handler member functions)
- Generate the application
- Build and run the application
- As desired, return to XVT-Design++ to edit the application, then regenerate it

**Tip:** Throughout this process, save the project often so you won't lose your work in case of a power outage or hardware failure.

## 3.2. The Hello Application

Our sample application is called “Hello”. It has the following features:

- The user can open any number of windows, by choosing the New command on the File menu. These windows display a message, chosen by the user, and can be moved, resized, and closed in the usual fashion for the native window system.
- The messages displayed in the windows can be changed by choosing commands on the Choices menu. Choosing the Other Choices command brings up a dialog that has several other message options, chosen with radio buttons.
- The user can select the font, size, and style for the message, using the Font menu.

## 3.3. Creating a New Project

In building the tutorial application, as in building any application with XVT-Design++, the first step is to create a new project file for the application. An XVT-Design++ project contains all of the resources and source code for your application's user interface.

While XVT-Design++ is running, create a new project by choosing New Project from the File menu. XVT-Design++ opens an Action Code Editor (ACE) window, as shown in Figure 3.

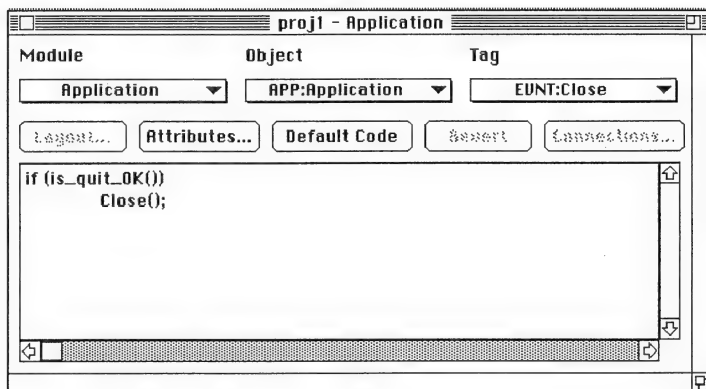


Figure 3. The Action Code Editor

We won't use the Action Code Editor right away, so close it by clicking its close box.

### 3.4. Creating a Menubar and Menu

Now that our project file is open, we'll create the menubar and menus for our application. Our finished menubar looks like Figure 4.



Figure 4. The Menubar for Our Sample Application

XVT-Design++ supplies the File, Edit, Font, and Style menus—they're referred to as the *standard menus*. We'll create the Choices menu, and its submenu.

To create this menubar, we'll follow these basic steps:

- Create a new menubar
- Create the Choices menu
- Add items to the Choices menu
- Create the submenu
- Add items to the submenu

### 3.4.1. The Menubar Editor

To create the new menubar, choose Menubar Editor from the Tools menu. The Menubar Editor is shown in Figure 5.

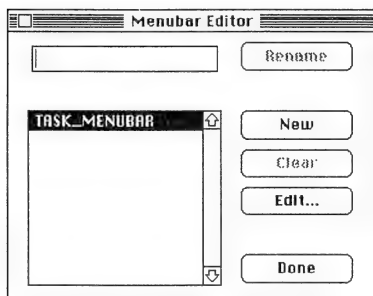


Figure 5. The Menubar Editor

The list box in the Menubar Editor shows all the menubars in the project. Notice that XVT-Design++ has automatically created one menubar, called TASK\_MENUBAR. TASK\_MENUBAR is the default menubar containing the four standard menus mentioned in the previous section. For now we can ignore this menubar; later we'll attach it to our task window.

#### 3.4.1.1. Creating a New Menubar

Click the New button to create a new menubar. XVT-Design++ adds a new menubar to the list box, with a default name of MENU\_BAR\_2. We'll change the default name to WIN\_MENUBAR. To do this, enter the name WIN\_MENUBAR in the edit field at the top of the Menubar Editor, and click the Rename button.

### 3.4.2. The Menu Editor

Next we'll add a new menu—the Choices menu—to the menubar we created. We'll use the XVT-Design++ Menu Editor.

In the list box, select our menubar, WIN\_MENUBAR. Click the Edit button in the Menubar Editor to bring up the Menu Editor. The Menu Editor is shown in Figure 6.

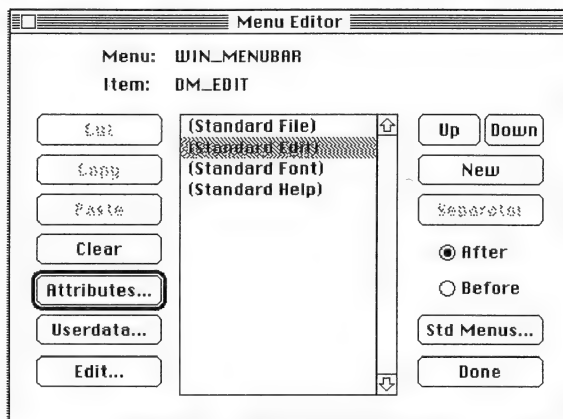


Figure 6. The Menu Editor

The name of the menubar appears at the top of the Menu Editor, to remind us which menu we're editing. The list box in the Menu Editor lists the names of the menus on the menubar, in left to right order (left at top, right at bottom in the list box). XVT-Design++ has already added the standard menus to our menubar; they are enclosed in parentheses.

In the Menu Editor, the New button creates new menus and adds them to the list. The Before and After radio buttons determine where in the list the new menu is placed—before or after the selected menu.

#### 3.4.2.1. Creating a New Menu

We want our Choices menu to appear to the right of the Edit menu on the menubar, so we'll insert it in the list after the Standard Edit menu.

The After radio button is already checked, so select the Standard Edit menu in the list box, and click the New button. A new menu appears in the list, with the default title "new item".

#### 3.4.2.2. Changing the Menu Title

One of the attributes of a menu is its title. We want our menu to have the title "Choices" rather than "new item", so we need to change the menu's title.

Click the Attributes button in the Menu Editor. The attributes dialog for our new menu appears. The title appears at the top of the dialog; click in the edit control and change the title to “Choices”. After you’ve changed the title, the dialog looks like Figure 7.

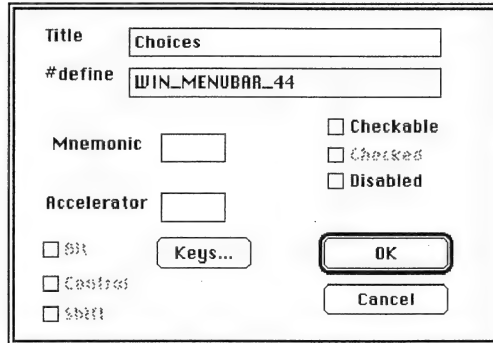


Figure 7. The Menu Attributes Dialog

Next, to add a mnemonic for the menu, place a “c” in that edit control. Notice that we could use this dialog to set other attributes, such as the menu’s `#define` string. For this menu, we’ll leave those unchanged. Click OK to dismiss the attributes dialog.

### 3.4.2.3. Adding Items to the Menu

So far we’ve created a new menubar, and added a menu to it. Now we need to add items to the menu. For our Choices menu, these items are “From Menu”, which has a submenu, and “From Dialog”.

Click the Edit button in the Menu Editor. This opens another Menu Editor window, which we’ll use to add the items to our Choices menu. The list of menu items is empty, since we haven’t added any items to this menu yet.

### Creating the First Item

To create the first item, click the New button. A new item is added to the list as shown in Figure 8.

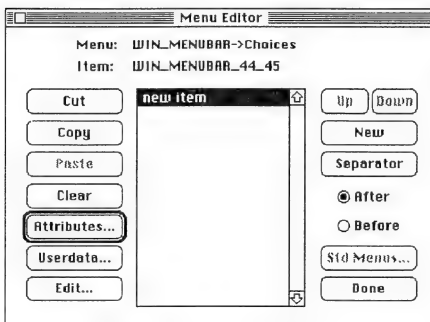


Figure 8. The Menu Editor Showing a New Item

We'll change the title of the new item, just as we did for the Choices menu itself. Click the Attributes button to open the attributes dialog for the new item. In the title field, change the title to "From Menu" as shown in Figure 9. Also add a mnemonic, "m".

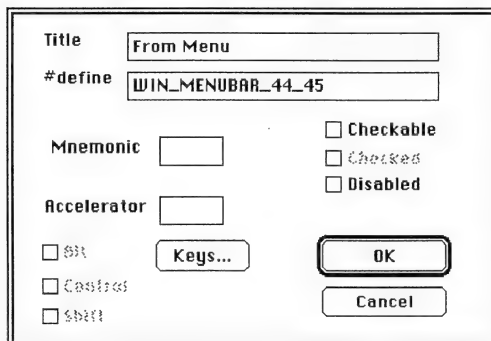


Figure 9. Changing the Menu Title to "From Menu"

Click OK to dismiss the attributes dialog.

### Creating the Second Item

Create another menu item, using the same procedure we followed to create the first item. Give it the title "From Dialog..." and the mnemonic "g".

### 3.4.2.4. Creating a Submenu

The final addition to our menubar is the submenu for the From Menu item. This submenu has two items, “Hello” and “Goodbye”. We want a check mark to appear next to these items when they are chosen while our application runs. We’ll build the submenu by adding items to the From Menu item, and setting their attributes, just as we did in the previous steps.

#### Creating the First Submenu Item

In the Menu Editor’s list box, select the From Menu item. Click the Edit button to open another Menu Editor window. Add a new item, as we did before, by clicking the New button, as shown in Figure 10.

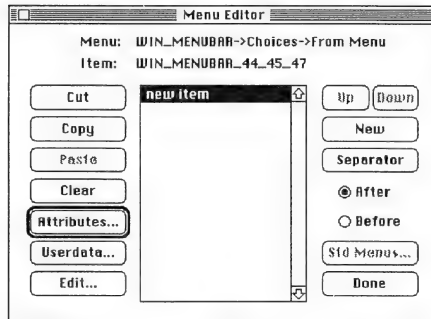


Figure 10. Adding a Submenu to Our “From Menu” Menu

Next, click Attributes to bring up an attributes dialog box for the item. Enter “Hello” in the Title field, “h” in the Mnemonic field, and “M\_HELLO” in the #define field. The #define string is used in your application’s source code to refer to the menu item, rather than using an integer ID number. Also, click the Checkable check box. See Figure 11.

The screenshot shows a dialog box titled 'Menu Editor'. It has two text input fields: 'Title' containing 'Hello' and '#define' containing 'M\_HELLO'. Below these are two more text input fields: 'Mnemonic' and 'Accelerator', both currently empty. To the right of these fields are three checkboxes: 'Checkable' (checked with an 'X'), 'Checked' (unchecked), and 'Disabled' (unchecked). At the bottom left, there are three checkboxes for keyboard modifiers: 'Alt' (unchecked), 'Control' (unchecked), and 'Shift' (unchecked). Next to these is a 'Keys...' button. At the bottom right are 'OK' and 'Cancel' buttons. The 'OK' button is highlighted with a thick border.

Figure 11. Changing the Menu Item Title to “Hello”

Click OK to dismiss the attributes dialog.

### Creating the Second Submenu Item

Create a second menu item called “Goodbye”, just as you created the Hello menu item. Set its mnemonic to “g” and #define string to “M\_GOODBYE”, then check its Checkable box. See Figure 12.

The screenshot shows a dialog box titled 'Menu Editor'. It has two text input fields: 'Title' containing 'Goodbye' and '#define' containing 'M\_GOODBYE'. Below these are two more text input fields: 'Mnemonic' and 'Accelerator', both currently empty. To the right of these fields are three checkboxes: 'Checkable' (checked with an 'X'), 'Checked' (unchecked), and 'Disabled' (unchecked). At the bottom left, there are three checkboxes for keyboard modifiers: 'Alt' (unchecked), 'Control' (unchecked), and 'Shift' (unchecked). Next to these is a 'Keys...' button. At the bottom right are 'OK' and 'Cancel' buttons. The 'OK' button is highlighted with a thick border.

Figure 12. Changing the Second Menu Item Title to “Goodbye”

We have now finished creating all of the menus and menu items for our application, so we can close all the Menu Editor windows. Click Done in each of the three Menu Editor windows. Then, in the initial Menubar Editor window, click Done again.

## 3.5. Saving the Project

Now is a good time to save the work you've done so far on this project. Choose Save Project from the File menu. Since this is the first time you've saved this project, a standard save-file dialog appears.

XVT-Design++ gives the project file a default name of "**proj1.prj**". Change the default name to "**hello.prj**". Choose a directory for this project and click the Save button. See Figure 13.

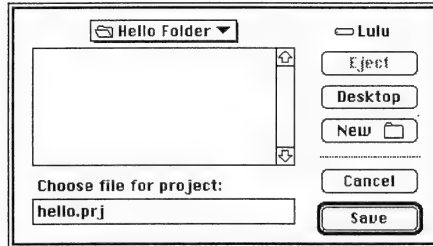


Figure 13. Saving Our "hello.prj" Project

**Tip:** As with any application, it's a good idea to save your file frequently, to avoid losing your work because of some unexpected disaster (such as a power outage).

## 3.6. Creating Containers

Next we'll create the containers—windows and dialogs—for our application. For our sample application, we need three containers:

- The document window for our application, which will display a message the user chooses
- A Choices dialog, with several controls, that will allow the user to choose one of several messages
- An "About box" dialog, which will display the name of our application

Each of these containers has several attributes, such as its name and size, that we'll need to change. First we'll create the document window and set its attributes. Then we'll create the Choices dialog and its controls. Finally, we'll create the About box.

### 3.6.1. Creating the Message Window

To create the window, choose New Window from the Window menu. XVT-Design++ creates a new window resource and opens a layout window for it as shown in Figure 14.

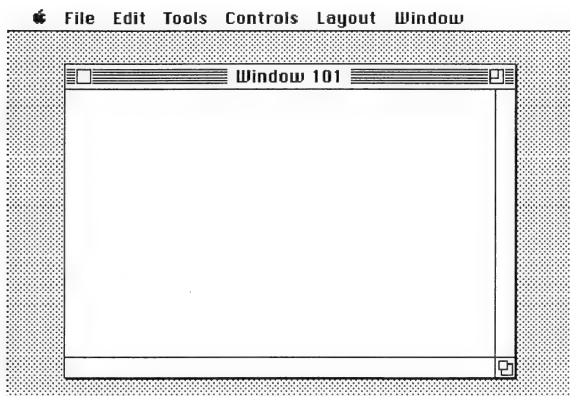


Figure 14. A Layout Window for a New Window

#### 3.6.1.1. Setting the Window's Attributes

Choose Attributes from the Edit menu to open the attributes dialog for the new window. (As a shortcut, you can double-click in the client area of the window to open its attributes dialog.) This dialog, shown in Figure 15, lets you change all of the attributes of the window, such as its title, size and location, and border style.

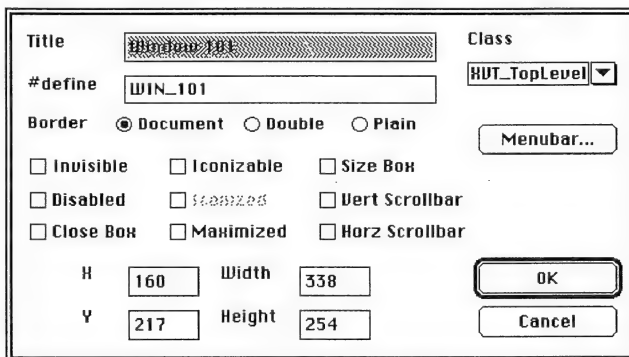


Figure 15. Setting Attributes for a New Window

XVT-Design++ gives the window a default title of “Window 101”. Change it to “Message” by clicking in the Title edit control and editing the string. Change the #define string from the default “WIN\_101” to “WIN\_MESSAGE”.

Click the check boxes labeled Close Box and Size Box. This tells XVT-Design++ to add these border controls to the window resource.

### 3.6.1.2. Associating the Window’s Menubar

We have already created a special menubar for the message window: the menubar we titled WIN\_MENUBAR. Now we need to associate this menubar with our application’s message window.

Click the Menubar button in the window’s attributes dialog. This opens a dialog with a list of all the menubars in your project. To associate our previously created menubar with the message window, click on WIN\_MENUBAR.

Click the OK button to dismiss this dialog. Finally, close the attributes dialog by clicking its OK button.

### 3.6.1.3. Adding a Push Button Control

Our message window will have one control, a push button. To create the button, choose Push Button from the Controls menu, and click in the lower section of the Message window. Double-click on the button to open its attributes dialog. See Figure 16.

The figure shows a dialog box titled "Setting Attributes for a Push Button". It contains the following elements:

- Title:** A text field containing "Push Button 1".
- #define:** A text field containing "WIN\_101\_PUSHBUTTON\_1".
- Justification:** A group box containing three radio buttons: "Native" (selected), "Left", and "Right".
- Standard Size:** A checked checkbox.
- Checked:** An unchecked checkbox.
- Default:** An unchecked checkbox.
- Disabled:** An unchecked checkbox.
- Cancel:** An unchecked checkbox.
- Read-only:** An unchecked checkbox.
- Invisible:** An unchecked checkbox.
- Multiple selections:** An unchecked checkbox.
- H:** A text field containing "112".
- Width:** A text field containing "110".
- V:** A text field containing "149".
- Height:** A text field containing "24".
- OK:** A button.
- Cancel:** A button.

Figure 16. Setting Attributes for a Push Button

The default title of the push button is “Push Button 1”; change it to “Custom String...”. Click the OK button to close the dialog.

You may find that the button is too small to contain the new name. If so, enlarge it by dragging the small rectangle near its lower-right corner. When you're done, the window should look like Figure 17.

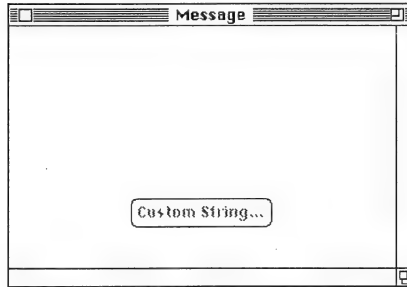


Figure 17. How the Push Button Looks in the Window

### 3.6.2. Creating the Other Choices Dialog

Now we'll create the Other Choices dialog. We'll first create an empty dialog and set its attributes. Then, we'll place some controls in the dialog, and set their attributes.

#### 3.6.2.1. Creating a New Dialog

To create the dialog, choose New Dialog from the Window menu. XVT-Design++ opens a layout window for the dialog.

**Note:** XVT-Design++ uses resizable windows to represent both windows and dialogs, so that you can easily change the size and position of the dialogs you create. When your finished application is running, a real native dialog will be used.

### 3.6.2.2. Setting the Dialog's Attributes

Double-click in the dialog layout window to open its attributes dialog, as shown in Figure 18.

The image shows a 'Setting Attributes for a Dialog' window. It contains the following fields and controls:

- Title:** A text field containing 'Dialog 102'.
- #define:** A text field containing 'DLG\_102'.
- Class:** A dropdown menu with a downward arrow.
- Type:** A group of four radio buttons: 'Modal', 'Invisible', 'Disabled', and 'Modeless'. The 'Modeless' button is selected.
- H:** A text field containing '160'.
- Width:** A text field containing '320'.
- Y:** A text field containing '100'.
- Height:** A text field containing '200'.
- Buttons:** 'OK' and 'Cancel' buttons are located at the bottom right.

Figure 18. Setting Attributes for a Dialog

Change the dialog's title to "Other Choices" and change its #define string to "DLG\_CHOICES". Click the Modal radio button to make the Choices dialog modal. Finally, click the OK button to close the attributes dialog.

### 3.6.2.3. Adding Radio Buttons

Now we'll add a group of radio buttons to the dialog. XVT-Design++ provides a handy method for creating multiple controls:

1. Before choosing the desired control from the Controls menu, press and hold the Shift key on your keyboard. Then choose the control.
2. Click in a layout window to create the control. Every time you click in the layout window, a control of this type will be created.
3. When you're done creating controls of this type, choose Pointer (or another control) from the Controls menu.

Use this method to place four radio buttons in the Choices dialog. Remember to choose Pointer from the Controls menu when you're finished creating controls. The dialog's layout window should look like Figure 19.

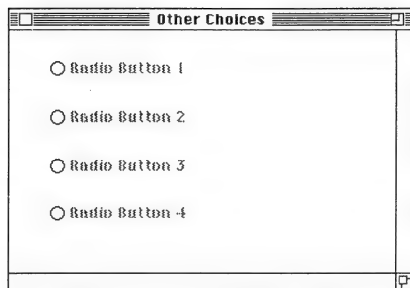


Figure 19. Adding Radio Buttons to the Dialog

#### 3.6.2.4. Changing the Radio Button Titles

Now we'll change the titles of the radio buttons to be the different messages that can be chosen in our application.

Click the first radio button to select it, and choose Attributes from the Edit menu (or simply double-click the first radio button) to bring up its attributes dialog. Change its title to "Have a nice day!", and click the OK button to dismiss the attributes dialog.

In a similar fashion, change the titles of the remaining radio buttons to "See ya later, alligator!", "Beam me up, Scotty!" and "Make it so!" (or whatever other messages strike your fancy).

**Note:** In the layout window, the titles of the buttons will probably be truncated at the right, so make the buttons larger by clicking on them and dragging the black rectangle at their lower-right corner. You may also need to change the size of the dialog itself to accommodate the controls; do this by resizing the dialog's layout window.

#### 3.6.2.5. Adding Push Buttons

We'll give the dialog two push buttons: an OK button and a Cancel button.

##### Creating the OK Button

Create a push button, and double-click it to open its attributes dialog. Change its title to "OK", and click the Default check box.

In a dialog, the button having the Default attribute appears with a thick border drawn around it, and it will recognize Return and Enter keystrokes. (For example, in the attributes dialog, the OK button is the default button.)

### Creating the Cancel Button

Create a second push button, and double-click it to open its attributes dialog. Change its title to “Cancel” and click the Cancel check box.

When you have finished setting the titles of the controls, the dialog should look like Figure 20.

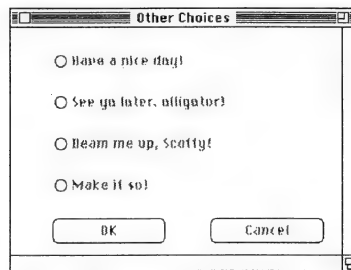


Figure 20. How Controls Look in the New Dialog

### 3.6.3. Creating an About Hello Dialog

The last container we’ll create is the About box. This will be a simple dialog with some text and one push button.

Choose New Dialog from the Containers menu to create the dialog. Open its attributes dialog, change its title to “About Hello”, its #define string to “DLG\_ABOUT”, and click its Modal radio button. Click OK to dismiss the dialog.

Next, add two static text controls and a push button to the About dialog. Change the title of the first static text control to “Hello version 1.0”, and the title of the second to “A simple application created with XVT-Design++”. Change the title of the push button to OK, and check its Default check box.

When you’re finished, the dialog should look like Figure 21.

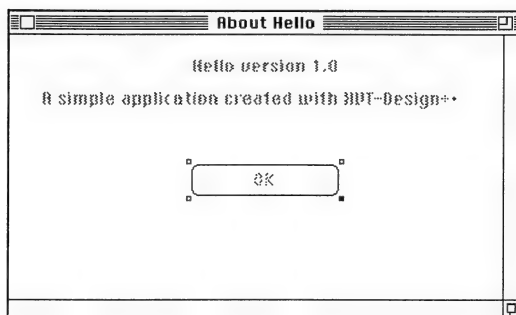


Figure 21. An About Box Dialog

## 3.7. Setting Application Attributes

So far we've concentrated on creating the individual user-interface objects of our Hello application, and setting their attributes. Now we need to set the attributes of the application itself.

Choose Project Attributes from the Edit menu to open the Project attributes dialog. In Project Attributes dialog, we will set the following four attributes:

### Task Menubar

The menubar attached to the Task Window. A list box shows the names of all the menubars in the project. The selected name is the Task Window's menubar. For our Hello application, this should be TASK\_MENUBAR.

### About Box

The dialog that is displayed when the application user brings up the About box. A list box shows the names of all the dialogs in the project. The selected name is the About box. Click on About Hello, the name of the dialog we created previously.

### Task Window Title

The title of the Task Window. The title is displayed in an edit control. Change it to "XVT-Design++ Tutorial".

### Document Prefix

A string put at the beginning of the titles of the application's document windows. It is displayed in an edit control. Change the Document Prefix to "Hello".

When you have finished setting the project attributes, the dialog should look like Figure 22.

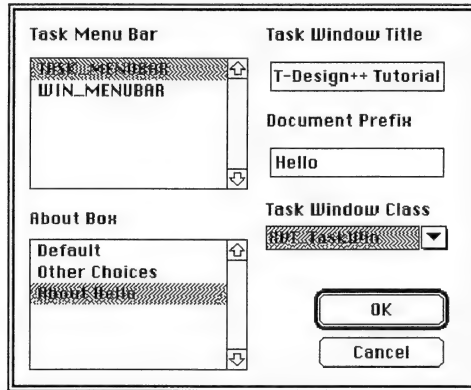


Figure 22. The Project Attributes Dialog

### 3.8. Setting Connections Between Interface Objects

Now that we've built all the user-interface objects for our Hello application, it's time to start building the program that will make these interface objects do something.

If you were creating this application without XVT-Design++, you would start writing C++ source code at this point. However, with connections and TestMode, XVT-Design++ lets you develop much of the functionality of your application's user interface *without* writing any code.

To start building our application's source code, we'll create connections by using the XVT-Design++ Action Code Editor.

Close any layout windows that you have left open, and choose Action Code Editor from the Tools menu. (You can leave layout windows open if you want, but your screen may not match the following illustrations if you do so.) The Action Code Editor looks like Figure 23.

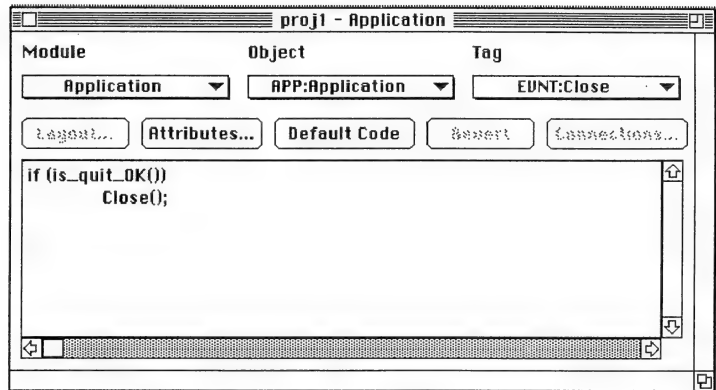


Figure 23. The Action Code Editor

### The ACE Context

Look at the three list buttons near the top of the ACE window, labeled Module, Object, and Tag. The settings of these list buttons constitute the *context* of the ACE—the unique combination of a module, one of the objects contained by the module, and one of the tags for the object.

The text-editing pane in the ACE always displays the code fragment for the context shown by the list buttons.

#### 3.8.1. Task Menubar Connections

When the user chooses New from the File menu of the Task Menubar, we want a new window to be created. So we'll set a connection for the Action tag of this menu item that opens our Message window.

To create this connection, first set the context of the Action Code Editor as follows:

- Set the Module list button to TASK\_MENUBAR
- Set the Object list button to ITEM:New
- Set the Tag list button to EVNT:Action

The list buttons in the ACE should look like Figure 24.

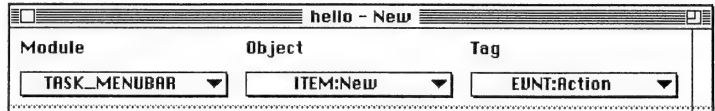


Figure 24. Setting the Context of the ACE

In the remainder of this chapter, we'll use the following format to represent the Action Code Editor's context:

TASK_MENUBAR	ITEM:New	EVENT:Action
--------------	----------	--------------

Click the Connection button to open the Connections dialog. Since we want this connection to open one of the containers we've constructed, click the radio button labeled Create User-defined Object.

A list button next to this radio button lists all the containers in the project. Set this list button to Message, the name of our application's window. The Connections dialog should look like Figure 25.

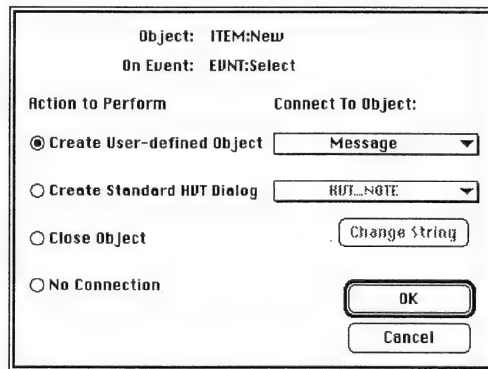


Figure 25. The Connections Dialog

Click the OK button to dismiss the dialog. Notice that XVT-Design++ adds the following code to the ACE's text-editing pane:

```
XVTAppWin101 *win = new XVTAppWin101;
win->Init( WIN_MESSAGE );
```

When executed, this code invokes our window.

## What Happens When You Create a Connection

When you create a connection, XVT-Design++ does two things:

- Records the connection in the project file, so that the connection will occur during TestMode
- Adds action code to the ACE's editing pane. This code is added to the module's source code file when XVT-Design++ generates your application's files, so that the connection will occur when the compiled application is executed

## Connections and Action Code

You can always add to or modify the action code that XVT-Design++ generates. Keep in mind that XVT-Design++ doesn't interpret or execute any of your application's action code. XVT-Design++ maintains connections separately from the code fragments. Hence you can have a connection with no code fragment (and vice versa).

### 3.8.2. Initial Message Window Connection

As a matter of style, we should create a Message window as the application is created. To do this, set the context in the Action Code Editor to:

Application	APP:Application	EVNT:Create
-------------	-----------------	-------------

Click the Connection button, and set the connection in the dialog just as you did for the Task window.

### 3.8.3. Message Window Menubar

We need to create two connections for our window's menubar, to accomplish the following:

- Open a new window when New is chosen from the File menu (the same as for the Task Window's menubar)
- Bring up the Other Choices dialog when From Dialog is chosen from the Choices menu

For the first connection, set the context in the Action Code Editor to:

WIN_MENUBAR	ITEM:New	EVNT:Action
-------------	----------	-------------

Click the Connection button, and set the connection in the dialog just as you did for the Task Window.

Notice that even though we have only created one window resource, at runtime our application can create any number of windows using this resource. We don't have to create a separate window resource for every window our application might create. In fact, we can let users of our application create as many windows as they like. The windows initially will have the same size and location, but can be moved independently and can display different messages.

For the second connection, set the context in the Action Code Editor to the following:

WIN_MENUBAR	ITEM:From Dialog	EVNT:Action
-------------	------------------	-------------

Open the Connections dialog, click the Create User-defined Object radio button, and choose Other Choices from the list button.

After you've completed these operations, the Connections dialog looks like the illustration in Figure 26.

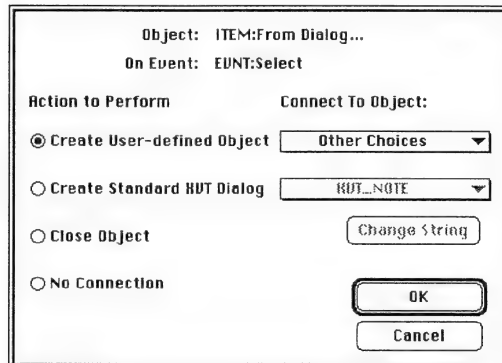


Figure 26. Connecting the "Other Choices" Dialog to the Choices Menu

Click the OK button to dismiss the Connections dialog.

### 3.8.4. Message Window Connections

Our application's window has one button. We'll use it to demonstrate the pre-defined dialogs available in XVT-Design++. We'll pretend that this button represents a feature in our application that has not yet been implemented.

Set the context in the ACE to:

Message	PB:Custom String	EVNT:Action
---------	------------------	-------------

This context tag corresponds to the event of the user clicking the button in our application's window.

Now click the Connection button, and check the Create Standard XVT Dialog radio button in the Connections dialog. Set the list button to XVT\_NOTE, and click the Change String button. In the small dialog that opens, type "Not Yet Implemented!" in the dialog's edit control. This is the message that will be displayed when the button in our application's window is clicked.

Click OK to dismiss the string dialog, then click OK in the Connections dialog to dismiss it.

### 3.8.5. Other Choices Dialog Connections

The connections for the Other Choices dialog are quite simple: when the user clicks either the OK or Cancel button, the dialog should go away. For both buttons, we'll create a connection that closes the dialog. (We must also handle the radio buttons in the dialog, but we'll take care of that later.)

To make the connection for the OK button, set the context to:

Other Choices	PB:OK	EVNT:Action
---------------	-------	-------------

Click the Connection button to open the Connections dialog. Click the Close Object radio button, then click OK to dismiss the dialog. See Figure 27.

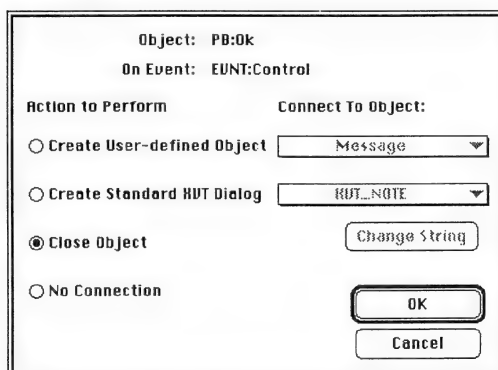


Figure 27. Creating a "Close Object" Connection

Now, when the OK button in our Other Choices dialog is clicked, its connection will close the dialog.

Set the same connection for the Cancel button, using this context:

Other Choices	PB:Cancel	EVNT:Action
---------------	-----------	-------------

### 3.8.6. About Hello Dialog Connection

The connection for the button in the About box is the same as that in the Other Choices dialog. When the user clicks the button, the dialog should go away.

To make the connection, first set the context to:

About Hello	PB:OK	EVNT:Action
-------------	-------	-------------

Then set the connection to Close Object.

## 3.9. Running TestMode

Now that we've created all of the user-interface objects for our application, and have defined connections between them, we'll use TestMode in XVT-Design++ to check our work. With TestMode, we can verify the appearance of our windows, dialogs, and menus, without compiling, linking, and running our application.

### Testing the Hello Application

To test the Hello application, choose Begin TestMode from the Tools menu. XVT-Design++ asks you to save your project if it hasn't been saved. Next, XVT-Design++ hides any open layout and Action Code Editor windows, and replaces its menubar with your application's Task Window menubar.

### Testing Connections

Let's exercise some of the connections we created. Choose New from the File menu, and our Message window appears (as shown in Figure 28). Notice that it has the correct menubar—the one we named WIN\_MENUBAR and associated with our Message window resource.

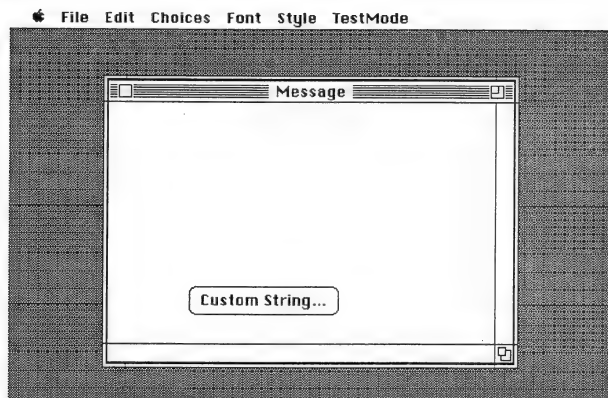


Figure 28. Message Window Showing a Menubar and Button

Try moving and resizing the window—it behaves as you would expect a GUI document window to behave. Also try creating more windows by choosing New from the File menu on either the Task Window’s menubar, or the Message window’s menubar. Either one creates a new window, since we defined the same connection for both.

If you click the Custom String button, a dialog with our sample XVT\_NOTE appears. Recall that we didn’t have to define the dialog explicitly—we asked XVT-Design++ to use a pre-defined dialog, and we gave it the string to display. Click OK to dismiss the dialog.

Try opening our Other Choices dialog, by choosing From Dialog from the Choices menu. At this stage, the radio buttons won’t do anything if you click them (we didn’t define any connections for them), but the OK and Cancel buttons will dismiss the dialog, as we intended.

Finally, choose End TestMode from the TestMode menu. We did not create this menu ourselves; XVT-Design++ added it to our application’s menubars to provide a way to leave TestMode. This menu is added only in TestMode. It will not appear in our final, compiled application.

### Making Adjustments to Interface Objects

You may have noticed that some of the user-interface objects needed minor adjustments. Perhaps you didn't like the location of the Messages window, or maybe the buttons in the Other Choices dialog were not arranged correctly.

You can fix these problems by re-opening the layout windows for the interface objects, adjusting the location of the resources, and re-entering TestMode to check your work.

XVT-Design++ lets you refine your application's appearance without the time-consuming process of editing and compiling resource and source code files, linking the modules, and running the application.

## 3.10. Attaching Action Code to Tags

While we have implemented much of our application's user-interface behavior with connections, we still need to define a few features in source code. Specifically, we must implement the following features:

- Displaying the appropriate message in the Message window, when the user chooses Hello or Goodbye from the Choices menu
- Placing a check mark next to the Hello or Goodbye menu item, after the user chooses one of them
- Displaying the appropriate message in the Message window, after the user clicks the OK button in the Other Choices dialog
- Changing the appearance of the message when the user chooses commands from the Font and Style menus

In accordance with object-oriented programming, one of our techniques to implement these features is to store information in the objects. This will be particularly true of the Message window, but also of the Other Choices dialog.

We will use the Action Code Editor to create action code for several tags. As you'll see, variables and member functions will be added to a class using `Class_Decl` tags. The object's variables will be defined within the Constructor and deleted within the Destructor. Create tags will be used to initialize the variables.

The Action code associated with many of our user interface objects—radio buttons and menu items—will reference the information stored in these variables as well as change or update their contents. Your own applications will undoubtedly utilize many of these same techniques.

**Tip:** You will probably find it useful to refer to the *XVT++ 2.0 Class Library Reference* as you complete this part of this tutorial.

### 3.10.1. Storing Information in the Message Window

Because we want our application's windows to display a message, we need a string variable to hold the message. We would also like to store the default message strings and we need to store pointers to two menu items. (Storing pointers to menu items makes it easier to check and uncheck them.) Since our application can have several windows open at once, and since each window can have a different message, we need to associate the variables with each window.

We'll add variables to the application's window class to store the information. We'll also add a member function that will handle the updating of the message and window. Set the context of the Action Code Editor to:

MessageWIN:	Message	SPCL:Class_Decl
-------------	---------	-----------------

and enter the following code in the ACE's editing pane:

```
// Data variables to store the window's messages and menu
// items and a member function to handle messages and updates.
char *Message, *Hello, *Goodbye;
XVT_MenuItem *HelloItem, *GoodbyeItem;
void SetMessage( const char *message );
```

To define the member function, set the context to:

Message	WIN:Message	SPCL:Bottom
---------	-------------	-------------

and enter the following code in the ACE's editing pane:

```
// Store the message to be displayed, make sure the menus
// are correct, and force the window to redraw its contents.
void XVTAppWin101::SetMessage( const char *msg )
{
    if (msg == NULL)
        return;
    delete [] Message;
    Message = new char [ strlen( msg ) + 1 ];
    strcpy( Message, msg );
    HelloItem->SetCheckedState( !strcmp( msg, Hello ) );
    GoodbyeItem->SetCheckedState( !strcmp( msg, Goodbye ) );
    Invalidate();
}
```

**Note:** XVTAppWin101 is the Message window class type. XVT-Design++ generates the name. The number in the name varies in accordance with when the window is created within XVT-Design++. By setting the context to:

Application	APP:Application	EVNT:Create
-------------	-----------------	-------------

the name that XVT-Design++ generates will be displayed as part of the code generated for the connection we had set earlier.

Within the window's Constructor, we will further specify the variables that we created using the Class\_Decl tag. To add code to the Constructor, set the context to:

Message	WIN:Message	SPCL:Constructor
---------	-------------	------------------

and enter the following code in the ACE's editing pane:

```
// Define standard messages that may appear in the window.
Hello = "Hello!";
Goodbye = "Goodbye!";
// Specify the initial values for the other added variables.
Message = new char[1];
HelloItem = GoodbyeItem = NULL;
```

Since we allocate memory when we construct the window, we'll need to free it when the window is destructed. Set the context of the ACE as follows:

Message	WIN:Message	SPCL:Destructor
---------	-------------	-----------------

and enter the following line of code:

```
// Free the string that stores the message in the window.
delete [] Message;
```

When a window is created, we want to do a number of things. First, for appearance's sake, we'll stagger the window. Next we'll use our added member function to specify an initial message. We'll also initialize the menubar by setting the Font menu to reflect the default font and enable the New menu item within the File menu.

Set the context of the Action Code Editor like this:

Message	WIN:Message	EVNT:Create
---------	-------------	-------------

and type the following code into the ACE's editing pane:

```
// Stagger new windows as they come up.
static int count = 0;
count++;
XVT_Rct boundary = GetInnerRect();
XVT_Pnt offset( 10*(count%6), 10*(count%6) );
boundary += offset;
SetInnerRect( boundary );
// Set the initial message in the window.
SetMessage( Hello );
// Initialize the Font menu.
XVT_DrawTools tools;
tools = DrawProtocol->GetDrawTools();
SetFontMenu(tools.GetFont());
// Make sure the New menu item is enabled.
XVT_MenuNode *enable_me;
enable_me = Menu->GetItem( M_FILE_NEW );
enable_me->SetEnabledState( TRUE );
```

Before we forget, we also need to enable the New menu item in the File menu of the Task Window's menubar. Set the context to:

Application	APP:Application	EVNT:Create
-------------	-----------------	-------------

and enter the following *before* the code that was generated by the connection we set earlier:

```
// Make sure the New menu item is enabled.
XVT_MenuNode *enable_me;
enable_me = Menu->GetItem( M_FILE_NEW );
enable_me->SetEnabledState( TRUE );
```

### 3.10.2. Displaying the Message

When a window receives an update event, its contents need to be redrawn. In our application, we'll erase the window and draw the string we placed there previously.

To add code to handle the update event, set the ACE's context to:

Message	WIN:Message	EVNT:Update
---------	-------------	-------------

XVT-Design++ has already added default code to erase the window's contents. Our code goes immediately *after* the default code. The code that XVT-Design++ generated is shown in italics:

```
// Clear the window to the default background color.
// Draw the text string that's stored in the window.
Clear();
XVT_Pnt point(20, 50);
DrawProtocol->DrawText(point, Message, -1);
```

### 3.10.3. Changing the Message with Menu Commands

When the user chooses Hello or Goodbye from the Choices menu, we want the corresponding string to appear in the Message window. So, for the Action tag of both of these menu items, we'll add code that uses the member function that we added to the Message window.

For the Hello item, set the context like this:

WIN_MENUBAR	ITEM:Hello	EVNT:Action
-------------	------------	-------------

and enter the following code:

```
// Send the new string to the window.
xdContainer->SetMessage( xdContainer->Hello );
```

The action code for the Goodbye item is almost identical:

WIN_MENUBAR	ITEM:Goodbye	EVNT:Action
-------------	--------------	-------------

```
// Send the new string to the window.
xdContainer->SetMessage( xdContainer->Goodbye );
```

As you may recall, as part of our SetMessage member function we check and uncheck menu items to reflect the window's current message. To enable this, we need to store a pointer to the menu item in the window. Code must be added to the Constructors of the menu items for this to happen. For the Hello item, set the context to:

WIN_MENUBAR	ITEM:Hello	SPCL:Constructor
-------------	------------	------------------

and enter the code that follows:

```
// Store the item in the window.
xdContainer->HelloItem = this;
```

Once again, we need to do the same thing to the Goodbye item. Set the context to:

WIN_MENUBAR	ITEM:Goodbye	SPCL:Constructor
-------------	--------------	------------------

and enter the code that follows:

```
// Store the item in the window.
xdContainer->GoodbyeItem = this;
```

### 3.10.4. Changing the Message with the Choices Dialog

Changing the message based on the user's interaction with the Other Choices dialog is slightly more complicated. We need a means for the dialog to communicate user actions to the window. Did they press the OK or Cancel button to close the dialog? Did they click a radio button?

We'll do this by storing a string variable in the dialog object. The controls' event handler member functions then handle the string according to the user's actions:

- When the user clicks a radio button, the event handler member function puts the appropriate message into the dialog's string message variable
- If the user clicks the OK button, the event handler member function sends the message string from the dialog object to the window object before returning

#### Storing the Message and Window Associated with the Dialog

To begin, we need a way to remember which window had focus when the Other Choices dialog was selected and a way to store a message when the user selects a radio button. We'll add variables to the application's dialog class to store both items. Set the context of the Action Code Editor to:

Other Choices	DLG:Other Choices	SPCL:Class_Decl
---------------	-------------------	-----------------

and enter the following code in the ACE's editing pane:

```
// A pointer to the window whose message can be changed.
XVTAppWin101 *win;
// The message from the currently selected radio button.
char *message;
```

When the user selects the Other Choices dialog, we will store the current window for future reference. Set the context of the Action Code Editor to:

WIN_MENUBAR	ITEM:From Dialog...	EVNT:Action
-------------	---------------------	-------------

and enter the following *between* the two lines of existing code:

```
// Store the window to send a message to.
dlg->win = xdContainer;
```

Within the Constructor for the dialog, we will set the message to NULL. Set the context to:

Other Choices	DLG:Other Choices	SPCL:Constructor
---------------	-------------------	------------------

and type the following code into the ACE's editing pane:

```
// Set the message value to NULL.
message = NULL;
```

When a dialog is created, we'll initialize the size of the message variable and set the default radio button to be the first one if the window's current message had not been specified using the dialog. This isn't simple or pretty, but it works.

Set the context of the Action Code Editor like this:

Other Choices	DLG:Other Choices	EVNT:Create
---------------	-------------------	-------------

and type the following code into the ACE's editing pane:

```
// Set the initial message string for the dialog and
// make sure an initial choice is made.
message = new char[64];
unsigned long len = 64L;
// Find the first radio button in the control list
xdctl = GetCtl( DLG_CHOICES_RADIOBUTTON_1 );
for (XVT_Control *ctl2 = GetFirstCtl();
     ctl2 != xdctl; ctl2=GetNextCtl())
;
// If the current message in the Window was from the Dialog
// check the radio button that corresponds to that selection.
for ( ; ctl2 != NULL; ctl2 = GetNextCtl() ) {
    ctl2->CastToRadioButton()->GetTitle(message, &len );
    if (!strcmp( message, win->Message)) {
        ctl2->CastToRadioButton()->SetCheckedState();
        break;
    }
}
```

```
// If the current message in the Window wasn't from the Dialog
// (i.e. doesn't match any of the radio buttons), select the
// first radio button as the default.
if (ctl2 == NULL) {
    xdctl->CastToRadioButton()->GetTitle(message, &len );
    xdctl->CastToRadioButton()->SetCheckedState();
}
```

Since the message variable was specified when we created the window, we'll need to delete it when the window is destroyed. Set the context of the ACE like this:

Other Choices	DLG:Other Choices	EVNT:Destroy
---------------	-------------------	--------------

and enter the following line of code:

```
// Free the string storage allocated within the Create tag.
delete [] message;
```

### Adding Code for the Radio Buttons

Each time a radio button is selected, we will store its title in the dialog's message variable. For the first radio button, set the context to:

Other Choices	RB:Have a nice day!	EVNT:Action
---------------	---------------------	-------------

and add the following code *after* the existing code in the edit pane:

```
// Store the button's title as the dialog's message.
unsigned long len = 64L;
GetTitle( xdContainer->message, &len );
```

The action code is exactly the same for each of the other radio buttons. For each one, set the context as shown, and enter the code that follows it.

**Tip:** You might find it convenient to use the Copy and Paste commands on the Edit menu to paste copies of text in each context.

Other Choices	RB:See ya later, alligator!	EVNT:Action
---------------	-----------------------------	-------------

```
// Store the button's title as the dialog's message.
unsigned long len = 64L;
GetTitle( xdContainer->message, &len );
```

Other Choices	RB:Beam me up, Scotty!	EVNT:Action
---------------	------------------------	-------------

```
// Store the button's title as the dialog's message.
unsigned long len = 64L;
GetTitle( xdContainer->message, &len );
```

Other Choices	RB:Make it so!	EVNT:Action
---------------	----------------	-------------

```
// Store the button's title as the dialog's message.
unsigned long len = 64L;
GetTitle( xdContainer->message, &len );
```

### Adding Code for the Push Buttons

Finally, we need to enter code for the push buttons. Both buttons already have action code for their tags, which XVT-Design++ inserted when we created connections for the buttons. The Cancel button needs no additional code, but the OK button needs code to send the dialog's message to the window's message. If a message has not been selected within the dialog, do not do a send:

Other Choices	PB:OK	EVNT:Action
---------------	-------	-------------

Add this text *before* the existing code:

```
// Dispatch a message to the Window to change its Message..
if (xdContainer->message)
    xdContainer->win->SetMessage( xdContainer->message );
```

### 3.10.5. Adding a #include

Because we reference the Message window type in the code for the Class\_Decl tag, we need to add a #include to the header of the Other Choices dialog source file. To do this, set the context to:

Other Choices	DLG:Other Choices	SPCL:Class Header
---------------	-------------------	-------------------

and enter the following code:

```
// Add an include for the Message window type declaration.
#include "helw101.h"
```

We'll see how this filename is generated in the next section.

## 3.11. Generating the Application

Now that we've created all of our application's resources, created connections between them, and added action code, we'll generate the source code files. XVT-Design++ will create complete source code, header, and resource files, as well as a platform-specific makefile to build the application.

### 3.11.1. Setting the Application Name

Before XVT-Design++ generates the files for our application, we need to tell it where to put the files. Choose Generated Files from the File menu. The dialog shown in Figure 29 opens.

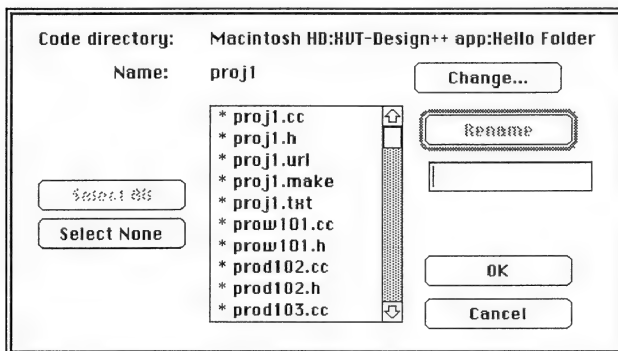


Figure 29. The Generated Files Dialog

Click the Change button. A standard save-file dialog appears. Navigate to the directory in which you want to place the generated code files, which will probably be the same directory that contains your project file.

Enter “hello” in the Name field. This is the name of the finished application. Click the Save button to dismiss the save-file dialog.

In the center of the Generated Files dialog, a list box shows the names of the files that XVT-Design++ will generate for your application. Notice that they have been changed to reflect the name you just gave to your application. See Figure 30.

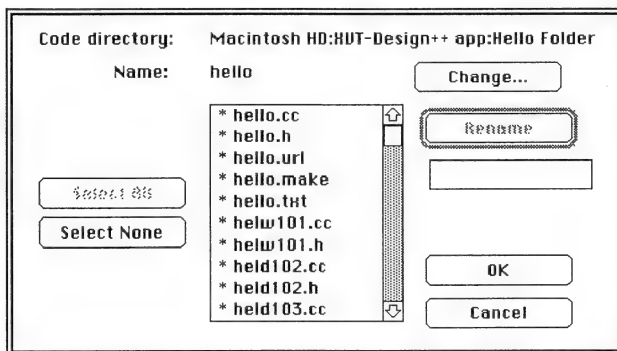


Figure 30. New Names for the Generated Files

Click OK to close this dialog.

### 3.11.2. Generating the Source Files

To generate the files, choose Generate Application from the File menu. XVT-Design++ now creates source code, header, and resource files in the directory you specified in the previous step. You may want to examine these files to see how XVT-Design++ incorporated the action code we created within its generated code.

## 3.12. Building and Running the Application

How you actually build the final application depends on your development environment:

- If you're using an environment with makefiles, XVT-Design++ will have created a makefile for your application. Use `make` (or whatever your make utility is called) to compile the application.
- If you're using an environment without makefiles, such as the Borland C++ IDE, use one of the project files in the **examples** directory in your XVT-Design++ installation.

Once you've compiled the application, run it and try out the features we constructed in this tutorial. Try changing the messages, first by choosing Hello and Goodbye from the Choices menu, and then by opening the Other Choices dialog and clicking different radio buttons.

# 4

---

## USAGE

This chapter describes the main features of XVT-Design++, and tells how to use them to build your application. The chapter covers the following topics:

- Project files
- The Action Code Editor
- Creating windows, dialogs, and controls
- Layout commands
- Interface object attributes
- Creation order
- The Menu Editor
- String resources
- Userdata strings
- TestMode
- Generating source code

### 4.1. Project Files

For every application you build with XVT-Design++, you begin by creating a new project file. Project files are the “documents” you create and modify with XVT-Design++. A project file contains all the resources and source code for your application’s user interface.

Projects are stored in binary files. These files are portable across all platforms on which XVT-Design++ runs. You can create a project file on one platform, then move it to another platform—without modification—to develop and refine your application on both platforms.

### 4.1.1. Creating New Projects

- ▼ To create a new project:

Choose New Project from the File menu. XVT-Design++ creates a new project and opens an Action Code Editor window for the project.

### 4.1.2. Working with Multiple Projects

You can work with more than one project at a time. The names of all open projects are listed at the bottom of the Edit menu.

Only one project is *active* at a given time. The active project has a check mark next to its name on the Edit menu. The windows for any inactive projects are hidden.

- ▼ To make a project active:

Choose the project's name from the Edit menu.

## 4.2. The Action Code Editor (ACE)

In XVT-Design++, the Action Code Editor (ACE) is the primary tool for creating and editing your application's source code. Using the ACE, you can

- Create and edit user action code
- Create and modify connections for TestMode
- Invoke other XVT-Design++ editors

This section describes how to use the ACE for each of these functions.

### 4.2.1. Invoking the ACE

You can invoke the ACE in several ways:

- From the Tools menu, choose Action Code Editor. You can use this method at any time, as long as a project is open
- From the Edit menu, choose Edit Code. This menu item is available only when the active window is a layout window for a window or dialog
- In a layout window, hold down the Shift key and double-click on a control, or in the layout window itself

## 4.2.2. ACE Controls

When opened in a new project, the ACE window looks like Figure 31.

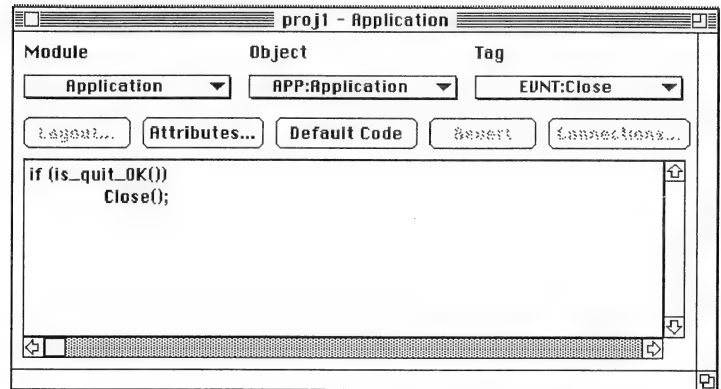


Figure 31. Controls in the Action Code Editor

The ACE window contains several controls: three list buttons for setting the editing context, a text-editing pane for examining and modifying source code, and several push buttons for other operations. The following sections describe the function of each control.

### 4.2.2.1. The Editing Context

In the ACE, *context* refers to a specific tag for a specific interface object. The context consists of three parts: the module that contains the object, the object itself, and the tag. Three list buttons in the ACE correspond to these parts (see below).

The titles of the list buttons always indicate the current context. Action code can be associated with each context. The action code is displayed in the text-editing pane of the ACE.

#### Module

The Module list button lists the titles of all the containers and menubars in the current project. An additional item, "Application," refers to the context for application events. The code for each module in the project is generated into a different C++ file.

## Object

The Object list button lists the titles of all the objects contained by the item specified by the Module list button:

- If the Module item is a window or dialog, the objects are the controls in that window or dialog
- If the Module item is a menubar, the objects are the titles of the menu items
- If the Module item is “Application,” there is only one object, also titled “Application”

Items in the Object list button have a prefix that indicates their type:

### Containers:

Prefix	Type
DLG	Dialog
WIN	Window

### Controls:

Prefix	Type
CB	Check Box
CC	Custom Control
ED	Editable Text
HS	Horizontal Scrollbar
LB	List Button
LE	List Edit
LX	List Box
PB	Push Button
RB	Radio Button
TE	Text Edit
TX	Static Text
VS	Vertical Scrollbar

## Tag

The Tag list button lists all the tags available for the item indicated by the Object list button. The items on this list vary depending on the

kind of object (control, menu item, etc.) in the context. There are two types of tags:

- *Event* tags have the prefix “EVNT:”
- *Special* tags have the prefix “SPCL:”

**See Also:** For a complete list of the tags associated with each interface object, see section 5.2.

#### 4.2.2.2. The Text Editing Pane

The center of the ACE window contains a rectangular pane for editing text. You’ll use this editor to create, modify, and examine all the action code fragments for your application. The code in this pane always corresponds to the current context of the ACE, as shown by the three list buttons.

If the code won’t fit in the text-editing pane, you can use the horizontal and vertical scrollbars to view it. You can change the size of the pane by resizing the ACE’s window.

To edit text in the ACE, you can use three Edit menu commands:

##### **Cut**

Removes the selected text from the editor and places it on the system clipboard.

##### **Copy**

Places a duplicate of the selected text onto the system clipboard.

##### **Paste**

Places the text from the system clipboard in the editor, at the insertion point. If text is selected in the editor, the clipboard text replaces it.

To change the font and size of the text in the ACE, choose the appropriate items from the Font menu. All text created with the editor is saved to disk when you save your project file. When XVT-Design++ generates the source files for your application, the text is placed in the appropriate files.

#### 4.2.2.3. Creating and Editing Connections

You create connections for TestMode in the ACE, using the Connections push button:

##### **Connections**

Brings up the Connection editing dialog (see Figure 32). This button is enabled only when the context is set to an object and tag that

allows a connection. In other words, it is enabled only when the context is a menu item's Action event, a push button's Action event, or the application's Create event.

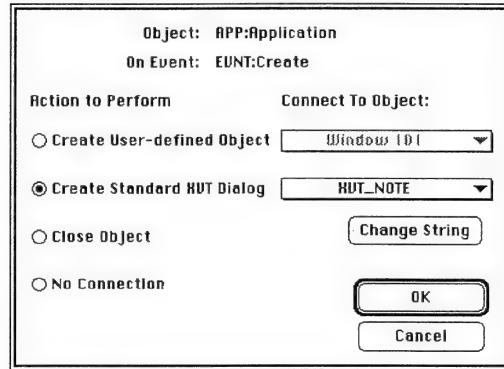


Figure 32. The Connection Dialog

The Connection dialog displays the interface object title and tag near the top to indicate the context for the connection. Four radio buttons on the left indicate the action that is executed when the event occurs.

### Create User-defined Object

If this button is checked, the connection creates one of your application's windows or dialogs. A list button, enabled only when this radio button is checked, indicates which one will be created. From the list, choose the name of the interface object that the connection will create.

### Create Standard XVT Dialog

If this button is checked, the connection invokes one of the predefined XVT dialogs. A list button, enabled only when this radio button is checked, shows the dialogs for the XVT\_GlobalAPI member functions Note, Error, Ask, and Message, along with the standard open-file and save-file dialogs. From the list, choose the name of the dialog that the connection will invoke.

### Change String

To enter the message that the dialog displays, click the Change String button. A small dialog opens; enter the message in the dialog's edit control and click the OK button. If you have selected the XVT\_ASK dialog for the connection, the dialog associated with

Change String has three additional edit controls for the titles of the push buttons.

### **Close Object**

If this button is checked, the action closes the window or dialog that contains the interface object. Typically this connection is used to dismiss a dialog or close a window when a button in that container is pushed.

### **No Connection**

If this button is checked, no connection will be made. The button is checked by default when the connection dialog is first opened for a context. To remove an existing connection, check this button.

**Note:** Checking this button does *not* remove the code previously inserted with a connection.

## **4.2.2.4. Using Other ACE Controls**

The ACE includes four other push buttons for performing various operations:

### **Layout**

Brings up the layout window for the context interface object. If the context is a menubar, the menubar editor opens. (See section 4.7.1.)

### **Attributes**

Opens the Attributes dialog for the context interface object.

### **Default Code**

XVT-Design++ supplies action code for some object/tag combinations. This “generic” code suggests what your application should do by default for the context. The code appears in the text-editing pane; you can edit it just as you would edit code you create yourself. After editing the code, you can restore the original generic code by clicking the Default Code button. The default code is added at the end of any existing text. The code is automatically selected so that you can move it easily with the Cut and Paste commands.

### **Revert**

Discards any changes you have made in the text editing pane. The pane reverts to the text that was present when the context was last selected.

## **4.3. Creating Windows, Dialogs, and Controls**

XVT-Design++ provides tools for creating windows, dialogs, and controls graphically and interactively. You create and adjust these resources directly on your screen, and XVT-Design++ generates the appropriate resource description files.

### **4.3.1. Creating Windows and Dialogs**

▼ To create a new window resource:

1. Choose New Window from the Window menu. XVT-Design++ opens a new layout window.
2. Move and resize the window to suit your needs. The size and location of the layout window represent the size and location of the window resource you have created.

▼ To create a new dialog resource:

1. Choose New Dialog from the Window menu. XVT-Design++ opens a new layout window.
2. Move and resize the dialog to suit your needs. The size and location of the layout window represent the size and location of the dialog resource you have created.

**Note:** XVT-Design++ uses document-style layout windows to represent windows and dialogs of all types. This lets you easily adjust the position and size of these resources. In TestMode, and in your compiled application, the windows and dialogs are rendered appropriately for their type.

### **4.3.2. Creating Controls**

▼ To create a control:

1. From the Controls menu, choose the control type.
2. Position the cursor in the upper left corner of the desired location.

3. Either click or drag the control into the desired size.  
If you click to create the control, it will be of the standard size for this type of control.
- ▼ To create multiple controls of the same type:
1. Press the Shift key while pulling down the Controls menu to choose a control.  
The cursor becomes a “+” to indicate visually that you are in multiple-control creation mode.
  2. Click or drag repeatedly to create several copies of the control.
  3. To exit from this mode, choose Pointer (or another control) from the Controls menu.
- ▼ To change the size of a control:
1. Click the control once to select it.
  2. Drag the small black rectangle near the lower-right corner of the control.

## 4.4. Layout Commands

The Layout menu includes various alignment and spacing commands to help you position controls in dialog boxes and windows. When you first create controls, you can position and/or size them manually by using the mouse or entering position and/or size values. Then you can fine-tune them using the layout commands.

### 4.4.1. Alignment

All alignment operations use the position of the first selected control as a reference point for lining up the other controls. For example, if you select push buttons 3, 2, and 1 (in that order), and then select Align Left from the Layout menu, push buttons 2 and 1 line up along the left border of push button 3, because it was the first control selected.

- ▼ To align controls:

From the Layout menu, choose one of the “Align” commands, or choose Even Horizontal Spacing, Even Vertical Spacing, or Make Same Size. See Figure 33.  
The controls align accordingly.

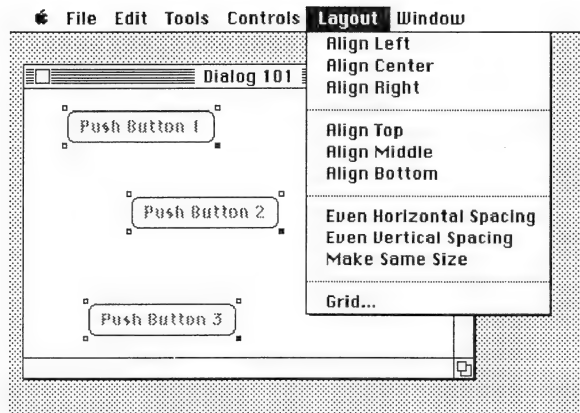


Figure 33. The Layout Menu

### **Align Left**

Aligns the selected controls along their left border. The position of the first control selected is the reference point for lining up the other controls.

### **Align Center**

Aligns the selected controls along their horizontal center by moving them left or right. The position of the first control selected is the reference point for lining up the other controls.

### **Align Right**

Aligns the selected controls along their right border. The position of the first control selected is the reference point for lining up the other controls.

### **Align Top**

Aligns the selected controls along their top border. The position of the first control selected is the reference point for lining up the other controls.

**Align Middle**

Aligns the selected controls along their vertical middle by moving them up or down. The position of the first control selected is the reference point for lining up the other controls.

**Align Bottom**

Aligns the selected controls along their bottom border. The position of the first control selected is the reference point for lining up the other controls.

**4.4.2. Spacing****Even Horizontal Spacing**

Equalizes the horizontal spacing between the right-most and left-most boundaries of the selected controls. If the controls would overlap, XVT-Design++ places them adjacent to each other horizontally, which increases the total distance between the right-most and left-most boundaries of the selected controls.

**Even Vertical Spacing**

Equalizes the vertical spacing between the top-most and bottom-most boundaries of the selected controls. If the controls would overlap, XVT-Design++ places them adjacent to each other vertically, which increases the total distance between the top-most and bottom-most boundaries of the selected controls.

**Note:** The Even Horizontal/Vertical Spacing options are enabled only when three or more controls are selected.

**Make Same Size**

Makes all selected controls the same size as the first control selected.

**4.4.3. Grid**

The Grid command lets you superimpose a grid on a selected window or dialog box, to help you position controls. You can determine the spacing of the grid, and choose whether controls snap to it.

- ▼ To superimpose a grid:
1. From the Layout menu, choose Grid.  
The Grid dialog box is displayed.
  2. Use the controls to set the grid.
  3. Click OK.

### Grid Spacing

Sets the horizontal and vertical spacing of the grid in pixels (default setting: 8 x 8) or in characters. You can use any size pixel-based grid, depending on how you like to lay out your controls. Or, for greater portability, you can use a character-based grid. (See “Tip,” later in this section.)

### Snap To Grid

Determines whether controls you create, move, or resize are automatically aligned to the layout grid. As you move a control, its position jumps to the nearest grid line intersection. As you change the size of a control, its right and bottom edges jump to the nearest grid lines. This option is independent of the Display Grid option; controls can snap to a grid even if it is not displayed.

### Display Grid

Determines whether the grid is displayed in the selected window or dialog box. This option is independent of the Snap To Grid option; you can display the grid without forcing controls to snap to the grid.

**Tip:** To ensure that your resources look good when you move your application to other platforms (especially a character-based platform), we recommend that you use the character-based grid. When you move your project from one platform to another, this grid changes size, but the grid spacing is always the size of a character of average width and height in the system font (the font used to draw labels in native controls). If you use a character-based grid, your controls will map cleanly to coordinate systems on other platforms, and thus be properly aligned and spaced.

## 4.5. Setting Interface Object Attributes

In XVT-Design++, the attributes for your application’s windows, dialog boxes, and controls are set with dialog boxes.

- ▼ To invoke the attributes dialog box for an interface object:
  1. Select the interface object (dialog box, window, or control).
  2. From the Edit menu, choose Attributes.

-OR-  
Double-click the interface object.
- ▼ To set attributes:
 

Enter text in edit controls.

- AND/OR -

Toggle check boxes on or off.

### 4.5.1. Common Attributes

The specific attributes that can be set vary depending on the type of interface object. However, there is a set of common attributes:

#### Title

Allows you to specify a particular title or name for an interface object. Each interface object's title field is automatically filled in with a system-defined default name.

#### #define name

Specifies a particular name for a resource ID associated with an interface object. These #define names for resource IDs are placed in a header file that is created when you invoke the Generate Application command. This header file is then included in the .url resource files and the .cc source code files that access the resources.

A #define name lets you symbolically refer to a particular resource within your application code. This field is automatically filled in with a system-defined default name.

**Caution:** If you edit the #define field, you must be sure that all #define names are unique. You cannot have two controls with identical #define names within a project.

#### X and Y Coordinate Locations

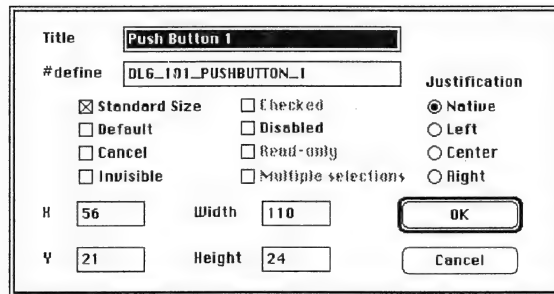
Specifies the positioning of an interface object. The X and Y coordinate values are specified in terms of pixels and refer to the position of the upper left-hand corner of the interface object. These values are set when you move the interface object in a layout window, or when you move the window itself. You can also type them in an attributes dialog.

## Height and Width Fields

Defines height and width values, in pixels, for a particular interface object. These values are set when you change the size of the interface object in a layout window, or when you change the size of the window itself. You can also type them in an attributes dialog.

The remaining attributes are interface-object-specific and are outlined in the following text according to the type of object.

### 4.5.2. Control Attributes



The image shows a dialog box titled "Attributes Dialog for Controls". It has a "Title" field with the text "Push Button 1" and a "#define" field with the text "DL6\_101\_PUSHBUTTON\_1". Below these are three columns of checkboxes: "Standard Size" (checked), "Default", "Cancel", "Invisible", "Checked", "Disabled", "Read-only", "Multiple selections", and "Justification" (radio buttons). The "Justification" column has radio buttons for "Native" (selected), "Left", "Center", and "Right". At the bottom, there are four input fields: "H" (56), "Width" (110), "V" (21), and "Height" (24). There are "OK" and "Cancel" buttons at the bottom right.

Title		Push Button 1			
#define		DL6_101_PUSHBUTTON_1			
<input checked="" type="checkbox"/> Standard Size	<input type="checkbox"/> Checked	Justification <input checked="" type="radio"/> Native <input type="radio"/> Left <input type="radio"/> Center <input type="radio"/> Right			
<input type="checkbox"/> Default	<input type="checkbox"/> Disabled				
<input type="checkbox"/> Cancel	<input type="checkbox"/> Read-only				
<input type="checkbox"/> Invisible	<input type="checkbox"/> Multiple selections				
H	56	Width	110	OK Cancel	
V	21	Height	24		

Figure 34. Attributes Dialog for Controls

## Standard Size

Sets a control to be the standard height, which is predefined for this type of control on this platform. On each platform, standard size is the native or "natural" size for a control of a certain type; in other words, it is the size that makes the control "look right."

If the standard size attribute is set, you can change only the width of a control, since the height is predefined. The exception to this is the vertical scrollbar control, where the width is predefined and you can change only the height.

To change the size in both dimensions, turn off standard size. Standard size for a control is set if you simply click to create the new control. If you drag to create the control, the standard size attribute is not set. The standard size attribute applies to all controls except list boxes and group boxes.

**Default**

Sets a push button control to respond to the default choice. This is the control that is automatically activated when the user presses Return; it is typically titled “OK”.

Only one push button can be the default. If you have set the Default attribute for one push button, then attempt to set the Default attribute for a second push button, the first push button’s Default attribute will be cleared. Setting the Default attribute also affects the creation order for controls, as explained later in this chapter. The Default attribute applies to push button controls in dialog boxes only.

**Cancel**

Sets a push button control to respond to the Cancel choice. This is the control that is automatically activated (on many platforms) when the user presses ESC; it is typically titled “Cancel”.

If you have set the Cancel attribute for one push button, and then attempt to set the Cancel attribute for a second push button, the first push button’s Cancel attribute will be cleared. The Cancel attribute applies to push button controls in dialog boxes only.

**Invisible**

Sets a control to be initially invisible. It can be made visible in the application by calling the XVT++ member function `XVT_Control::SetVisibleState`.

**Checked**

Places a system-generated check mark in a control. The checked attribute applies to check box and radio button controls only.

**Disabled**

Sets a control to be initially disabled. The control is grayed out so that it is not selectable. It can be enabled in the application by calling the XVT++ member function `XVT_Control::SetEnabledState`.

**Read-only**

Sets a control to be read-only (not selectable). Applies to list boxes and text edit objects only.

### Multiple selections

Allows multiple items listed within a control to be selected. The multiple selections attribute applies to list box controls only.

### Justification

Sets whether a control's text label will be left-, center-, or right-justified (when possible for a particular toolkit). This attribute defaults to native justification, which tells XVT++ to use whatever justification is used by default by the native platform. The justification attribute applies to all "labeled" controls: push buttons, check boxes, radio buttons, group boxes, and static text.

#### 4.5.2.1. Custom Controls

The dialog box titled 'Custom Controls Dialog' contains the following fields and controls:

- Title:** A text input field containing 'Custom Control 1'.
- #define:** A text input field containing 'WIN\_101\_CUSTOM\_1'.
- Class:** A dropdown menu with 'custom' selected.
- H:** A text input field containing '70'.
- Width:** A text input field containing '160'.
- Y:** A text input field containing '80'.
- Height:** A text input field containing '120'.
- Buttons:** 'OK' and 'Cancel' buttons.

Figure 35. Custom Controls Dialog

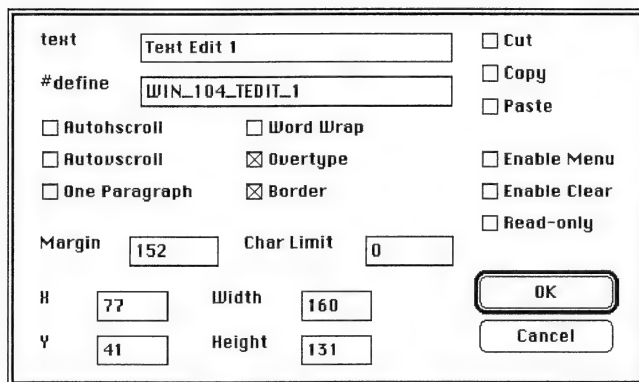
Custom controls do not have the same attributes as standard controls in XVT-Design++. Custom controls have only the attributes common to all interface objects (title, #define name, location, and size), and one additional attribute that specifies the control's class.

### Class

Specifies the class name for the control. By default, XVT-Design++ sets the name to "custom". To change the name, type one in the Class edit field, or select a name from the field's list.

#### 4.5.3. Text Edit Attributes

The following attributes apply to text edit objects only. To learn more about text edit objects, see the *Portability Toolkit Manual*.



The dialog box is titled 'Attributes Dialog for Text Edit Objects'. It contains the following fields and options:

- text**: Text Edit 1
- #define**: WIN\_104\_TEDIT\_1
- Options**:
  - ☐ Autohscroll
  - ☐ Autovscroll
  - ☐ One Paragraph
  - ☐ Word Wrap
  - ☒ Overtime
  - ☒ Border
  - ☐ Cut
  - ☐ Copy
  - ☐ Paste
  - ☐ Enable Menu
  - ☐ Enable Clear
  - ☐ Read-only
- Margin**: 152
- Char Limit**: 0
- H**: 77
- Width**: 160
- Y**: 41
- Height**: 131
- Buttons**: OK, Cancel

Figure 36. Attributes Dialog for Text Edit Objects

### Autohscroll

Enables automatic horizontal scrolling of the text edit object when the user drags the mouse outside of the view rectangle.

### Autovscroll

Enables automatic vertical scrolling of the text edit object when the user drags the mouse outside of the view rectangle.

### One Paragraph

Limits the entered text to one paragraph. A paragraph is terminated with a carriage return. When this is set, the paragraph is limited to the number of characters entered in the Char Limit field.

### Word Wrap

Keeps words together and wraps them to the next line when there is not enough room on the first line (as determined by the number of pixels specified in the Margin field).

### Overtime

Determines whether the text edit object is initially in overtime mode. In this mode, typed characters overwrite existing text. If this attribute is not set, the text edit object is set to insert mode.

**Border**

Places a border around the text edit object.

**Cut**

Allows users to cut text from the text edit object.

**Copy**

Allows users to copy text from the text edit object.

**Paste**

Allows users to paste text into the text edit object.

**Enable Menu**

Determines whether the text edit system will enable and disable the commands on the Edit menu of the window that contains the text edit object. Do not check this attribute if the window's menubar does not have an Edit menu.

**Enable Clear**

Causes the Clear menu item to be always enabled when the text edit object is active.

**Read-only**

Sets the text in the text edit object to be read-only (not selectable).

**Margin**

Sets the maximum number of pixels allowed per line when Word Wrap is enabled, thereby determining the right margin.

**Char Limit**

Sets the limit for the number of characters that can be entered in the text edit object, if the One Paragraph attribute has been set.

**Note:** The Change Text Edit Attributes dialog lets you set the "Title" of the text edit object whose attributes you are changing in XVT-Design++. If you change the text in the title and click the OK button of this dialog, the altered text appears at the top of the text edit object you are editing. In the generated application, the text edit object initially contains the text you entered as the "Title" of the text edit object.

## 4.5.4. Dialog Box Attributes

The screenshot shows a dialog box titled 'Attributes Dialog for Dialog Boxes'. It has several input fields and checkboxes. The 'Title' field contains 'Dialog 107'. The '#define' field contains 'DLG\_107'. The 'Class' field is a dropdown menu. The 'Type' section has three radio buttons: 'Modal', 'Modeless' (which is selected), and 'Invisible'. There are also checkboxes for 'Invisible' and 'Disabled'. The 'H' field is '160', 'Width' is '320', 'Y' is '100', and 'Height' is '200'. At the bottom right are 'OK' and 'Cancel' buttons.

Figure 37. Attributes Dialog for Dialog Boxes

**Note:** In XVT-Design++, dialog layout is done in a window. In your application and in TestMode, however, the layout will appear in a dialog.

### Modal/Modeless

This control sets the type of the dialog box being created as either modal or modeless. It does not affect how dialog boxes are presented in XVT-Design++ during layout, but is reflected in the code that XVT-Design++ generates and in TestMode.

### Invisible

Sets the dialog to be initially invisible. It can be made visible in the application by calling the XVT++ member function `XVT_Dialog::SetVisibleState`.

### Disabled

Sets the dialog to be initially disabled. It can be enabled in the application by calling the XVT++ member function `XVT_Dialog::SetEnabledState`.

### Class

Specifies a new class name for a dialog. If you enter a name in a dialog's class edit field, or select a name from the class list, XVT-Design++ generates a new class definition derived from this class.

You can write member functions for the new class to supplement existing member functions of the superclass. The member functions

for the new class will invoke the member functions of the superclass before they themselves are executed.

### 4.5.5. Window Attributes

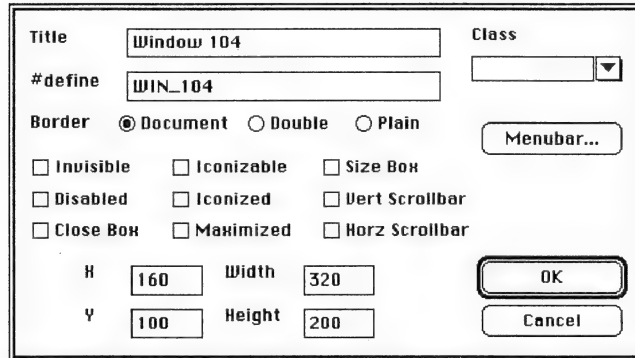
A screenshot of the 'Attributes Dialog for Windows' in XVT-Design++. The dialog has a title bar and contains several sections. At the top, there are text fields for 'Title' (containing 'Window 104') and '#define' (containing 'WIN\_104'). To the right of these is a 'Class' dropdown menu. Below the text fields is a 'Border' section with three radio buttons: 'Document' (selected), 'Double', and 'Plain'. To the right of the radio buttons is a 'Menubar...' button. Below the 'Border' section is a grid of checkboxes for various window attributes: 'Invisible', 'Disabled', 'Close Box', 'Iconizable', 'Iconized', 'Maximized', 'Size Box', 'Vert Scrollbar', and 'Horz Scrollbar'. At the bottom, there are text fields for 'H' (160), 'Y' (100), 'Width' (320), and 'Height' (200). On the right side of the dialog are 'OK' and 'Cancel' buttons.

Figure 38. Attributes Dialog for Windows

#### 4.5.5.1. Attributes that Do Not Affect XVT-Design++ Appearance

The following attributes do not affect the appearance of a window as displayed in XVT-Design++ during layout. However, they are reflected in the code generated by XVT-Design++ and in TestMode.

##### Border

Sets the window's type. There are three types available: Plain border, Double Border, and Document.

##### Class

Specifies the class name for a window. If you enter a name in a window's class edit field, or select a name from the class list, XVT-Design++ generates a new class definition derived from this class.

You can write member functions for the new class to supplement the existing member functions of the superclass. The member functions for the new class will invoke the member functions of the superclass before they themselves are executed.

**Invisible**

Sets a window to be initially invisible. It can be made visible in the application by calling the XVT++ member function `XVT_ChildBase::SetVisibleState`.

**Disabled**

Sets a window to be initially disabled. It can be enabled by the application by calling the XVT++ member function `XVT_ChildBase::SetEnabledState`.

**4.5.5.2. Attributes that Affect Only Document-Type Windows**

The following attributes affect only windows with their Type attribute set to Document.

**Close Box**

Determines whether a window's decoration includes a close box.

**Iconizable**

Determines whether a window can be iconized. This affects only the Win, PM, and Motif platforms.

**Iconized**

Sets a window to be initially iconized (minimized). This affects only the Win, PM, and Motif platforms. This attribute cannot be set if Maximized is set (see below).

**Maximized**

Sets a window to be initially maximized. Motif platforms ignore this setting. This attribute cannot be set if Iconized is set (see above).

**Size Box**

Determines whether a window's decoration includes a size box.

**Vertical Scrollbar**

Determines whether a window includes a vertical scrollbar.

**Horizontal Scrollbar**

Determines whether a window includes a horizontal scrollbar.

## 4.6. Creation Order

On the Edit menu, the Creation Order command lets you specify the creation order for controls within the currently active dialog box or window. When a user navigates an application by pressing keys (assuming the platform supports such keyboard navigation), the creation order for controls determines the order in which the controls are traversed.

- ▼ To view or edit the creation order for controls:
1. Select a window or dialog box.
  2. From the Edit menu, choose Creation Order.  
A dialog box shows the control names and their current traversal order; see Figure 39.

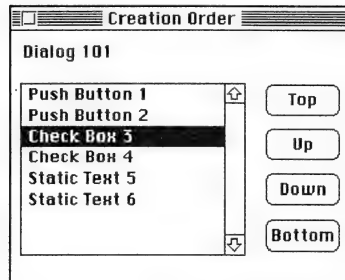


Figure 39. The Creation Order Dialog

The current traversal order reflects the order in which you created the controls, except for two cases:

- Any push button control with the Default attribute automatically appears in the first position
- Any push button with the Cancel attribute automatically appears in the second position

- ▼ To rearrange the current order:
1. Select the name of the control to be moved.
  2. Click the Up, Down, Top, or Bottom button to move the control to a new position.

### **Up, Down**

Moves the selected control name one position up or down.

### **Top, Bottom**

Moves the selected control name to the top or bottom of the list.

**Note:** If any push buttons have the Default or Cancel attribute, a control name cannot move to the top of the list.

## 4.7. The Menu Editor

With XVT-Design++, you can design multiple menubars for your application. Each menubar can have hierarchical menus that descend from it. To help you understand menus in XVT++, here are some basic definitions:

### Menubar

A menubar is the “root” of the menu hierarchy tree. To design menus, you must start with a menubar. A menubar, which consists of a list of menus, is visually represented by a row of names across the top of a screen or window.

### Menu (also called a **pull-down menu**)

Menus appear horizontally across a menubar. When you click on a menu (or select it via a keyboard mnemonic), it “pulls down” a vertical list of items for you to choose from. A menu can contain submenus.

### Menu item

Menu items appear on a menu or submenu. A menu item can be a “leaf” of the menu tree, in which case it will invoke the member function (e.g., `XVT_MenuItem::e_action`) associated with the menu item. Or, it can be another submenu whose contents are displayed when the user drags the mouse to this item.

### Hierarchical menu

A hierarchical menu has one or more submenus. Such a menu/submenu arrangement is hierarchical because it can contain several nested levels of menus.

### Submenu

A submenu is just like a menu, except that it can appear anywhere in your menu hierarchy. When a submenu appears as an item on a menu (or submenu), some graphical indication—such as an arrow—is used to show that the menu hierarchy extends below this item. When the user pulls down a menu and moves the mouse to a submenu, the list of menu items for that submenu appears. See Figure 40.



Figure 40. Hierarchical Menu with Submenu

In the illustration above, the menubar consists of the File, Edit, Choices, Font, and Style menus. The Choices menu is a hierarchical menu; its submenu is titled “From Menu”. The From Menu submenu has two menu items, titled “Hello” and “Goodbye”.

#### 4.7.1. Menubar Editor

You can create new menubars and edit their menu hierarchies by using the Menubar Editor.

- ▼ To invoke the Menubar Editor:

From the Tools menu, select Menubar Editor.

The Menubar Editor dialog appears, as shown in Figure 41.



Figure 41. The Menubar Editor

The Menubar Editor consists of a dialog containing a list box, an edit control, and several push buttons. The list box shows the names of all menubars in your project. To select a menubar, click on its name. The push buttons apply different actions to the selected menubar.

The Menubar Editor dialog contains the following controls:

##### Rename

Changes a menubar’s #define name. To do this, select the menubar in the list box and type the new name in the edit control. Then click the Rename button.

**New**

Creates a new menubar. The name of the new menubar is added to the list.

**Clear**

Deletes the selected menubar. The menubar is permanently removed from the project, and it is not copied to the clipboard.

**Edit**

Invokes the Menu Editor for this menubar; see section 4.7.2.

**Done**

Dismisses the Menubar Editor.

**4.7.2. Menu Editor**

When you invoke the Menu Editor for a menubar, a list box appears containing the four standard menus: File, Edit, Font, and Help. The menus are listed in left-to-right order. See Figure 42.

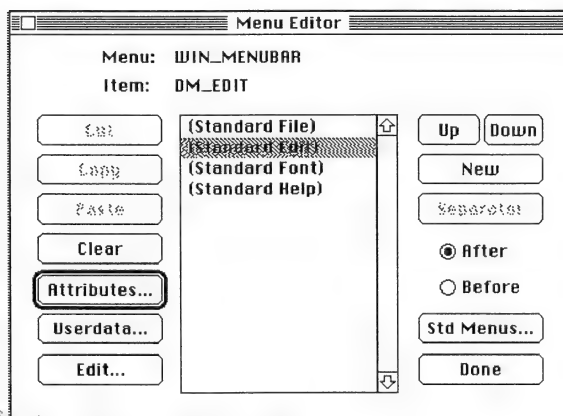


Figure 42. Menubar Editor Showing Standard Menus

**Cut**

Deletes the selected menu item and puts it on the clipboard.

### Copy

Copies the selected menu item to the clipboard.

### Paste

Inserts the contents of the clipboard either before or after the selected menu item (according to the “Before/After” setting).

### Clear

Deletes the selected menu item without putting it on the clipboard, thereby permanently deleting it.

### Attributes

Invokes the Menu Attributes dialog, where you can set various attributes for a menu, menu item, or submenu; see section 4.7.3.

### Userdata

Invokes the Userdata Editor, where you can associate up to six strings of arbitrary text with a menu, menu item, or submenu. Userdata strings are retrieved with the XVT\_GlobalAPI member function `GetMenuUserData`.

### Edit

Invokes the Menu Editor again, so you can descend to the next level below the selected menu item. See Figure 43.

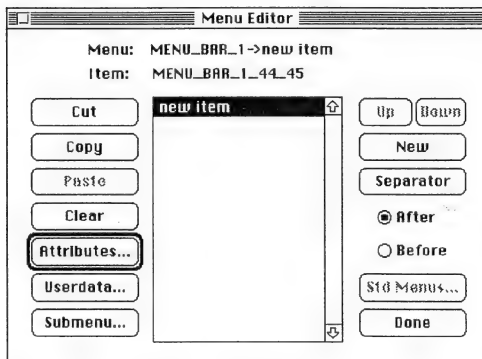


Figure 43. Editing a Submenu

### Up/Down

Moves the selected menu item either one position up, or one position down in the current menu. Note that you cannot position any items in front of the standard File or standard Edit menus, and you cannot position any items after the standard Font or standard Help menus.

### New

Creates a new menu item, either before or after the currently selected menu item (according to the “Before/After” setting). See Figure 44.

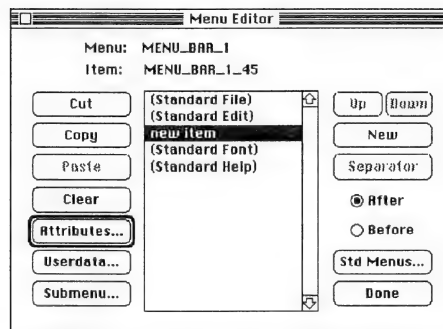


Figure 44. Menu Editor Showing a New Item

### Separator

Inserts a separator bar either before or after the currently selected menu item (according to the “Before/After” setting). You can insert separators only into menus and submenus; you cannot insert them on the menubar.

### Before/After

Determines whether new items, pasted items, and separators are inserted before or after the selected menu item.

### Std Menus

Brings up a dialog where you can select which standard menus you want. The standard menus (File, Edit, Font, Help) can be selected only for a menubar, not its submenus. See Figure 45.

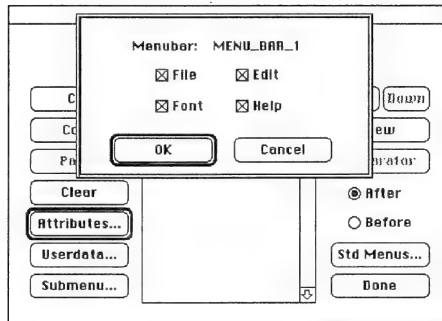


Figure 45. Dialog for Selecting Standard Menus

### Done

Dismisses the Menu Editor dialog. You can also dismiss the dialog by clicking its close box.

## 4.7.3. Menu Attributes

When you click Attributes for a menu item, the Menu Attributes dialog is displayed as shown in Figure 46.

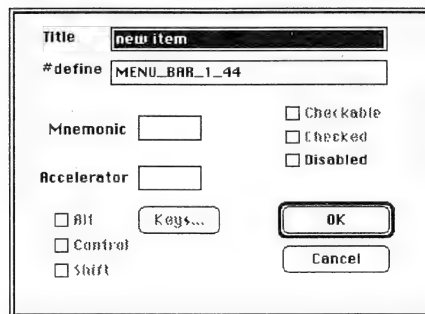


Figure 46. The Menu Attributes Dialog

The Menu Attributes dialog contains the following controls:

### Title

The text for this menu item.

**#define**

The #define name for this menu item.

**Mnemonic**

Specifies the one-character mnemonic you wish to associate with this menu item. At runtime, the mnemonic can be used on those platforms that support keyboard navigation of menus. The mnemonic character must be one of the characters in the title for this menu item.

**Accelerator**

Specifies the one-character accelerator key that you wish to associate with this menu item. At runtime, the accelerator can be used as a substitute for selecting the menu item with the mouse.

**Keys**

Brings up the Accelerator Keys dialog, where you can specify a function key as the accelerator for this menu item. When you choose one of the keys, its full name is inserted in the Accelerator edit field.

**Alt/Control/Shift**

If you have selected an accelerator for this menu item, you can specify the modifier keys (Alt, Control, or Shift) to be used with the accelerator key.

**Checkable**

Specifies that this menu item can be checked. The member function `XVT_MenuItem::SetCheckedState` can be used to check or uncheck an item that is checkable.

**Checked**

Specifies that this menu item should initially be checked.

**Disabled**

Specifies that this menu item should initially be disabled. The item can be enabled by the application using the XVT++ member function `XVT_MenuNode::SetEnabledState`.

**OK**

Dismisses the Menu Attributes dialog, saving all changes.

**Cancel**

Dismisses the Menu Attributes dialog, discarding all changes.

## 4.8. String Resources

### 4.8.1. Strings

In XVT-Design++, you can create and manipulate strings and string lists, to be used as resources from within an XVT++ program. The advantage of string resources is that you can maintain strings outside your executable program. As a result, you can modify them without having to recompile the program.

Strings and string lists are useful for adding text to your application that would not otherwise be in its resources. For instance, you could use a string list to initialize a list box or list button. String resources are also useful for holding text that may change when the application runs. As an example, you could have a button with the title “Find”, that changes to “Find Again” at some point during execution. The second string, “Find Again”, could be stored as a string resource.

▼ To create a string:

1. From the Tools menu, choose Strings Editor.  
The Strings dialog box appears as shown in Figure 47.

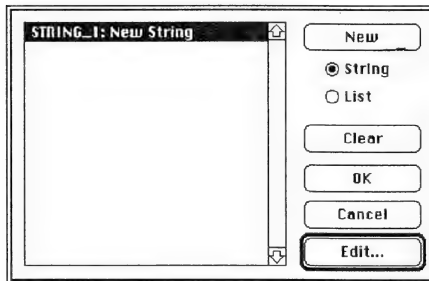


Figure 47. The Strings Dialog

2. Click New.  
A new string is created, whose contents are initially “New String”.

3. Click Edit (or double-click the string in the list box).  
The String Edit dialog box appears as shown in Figure 48, in which you can change both the string and the #define name.

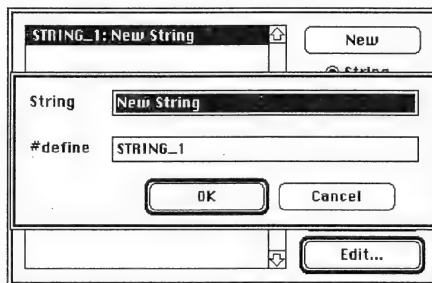


Figure 48. The String Edit Dialog

**Example:** If you change the string to “alignrt”, change the #define name to “STR\_ALIGNRT”, then click OK, the updated string appears in the Strings list box. In your XVT++ application, you would need to add a call to `XVT_GlobalAPI::GetResString` to retrieve the string at runtime. Its function prototype is:

```
BOOLEAN
GetResString(
    char*      buffer,
    long       rid,
    unsigned long* len )
```

For the second argument, simply use the #define name for the string you want (in the above example, you would use `STR_ALIGNRT`). You must allocate a buffer large enough to hold the string and pass the address of the buffer, as well as the buffer’s length, as the first and third arguments.

**See Also:** For more information about `GetResString`, see the *XVT++ Class Library Reference*.

#### 4.8.2. String Lists

String lists are useful if you have a list of names or labels that you would like to enumerate outside of your application. Let’s say you had a list containing the names of all the states in the United States. In your application, you could retrieve the list as a whole, and populate a list box with the resulting list.

▼ To create a string list:

1. From the Tools menu, choose Strings Editor.
2. In the Strings dialog, click the List radio button.
3. Click the New button.  
A new string list is created.

Notice that all you see is the `#define` name for the list. The strings themselves are not visible.

▼ To edit the elements in the string list:

Click the Edit button.

-OR-

Double-click the `#define` name for the string list in the list box.

The String List dialog box appears as shown in Figure 49.

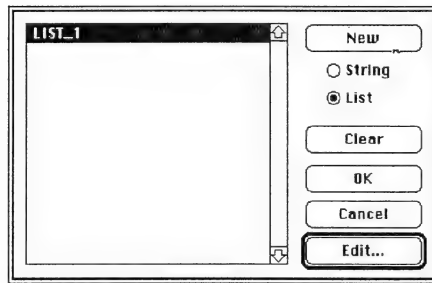


Figure 49. The String List Dialog

**Example:** In the String List dialog box, the `#define` name, which you can change, appears at the top. In our example, you might change this to be `SL_USA` (since the string list will hold all the states in the United States).

When you click New, a new entry with the contents “New String” is created. To change this, either click Edit or double-click the item in the list box. The String List Edit dialog box appears, in which you can change the contents of the string (perhaps to something like “Arizona”), and click OK.

Click New again (which creates another “New String”), and change it to “Alabama”. Clicking New one more time creates another string that you could change to “Georgia”.

To arrange your strings in ascending alphabetical order, you can reorder your string list. To reorder an item, select the one you would like to reorder (in our example, “Alabama”), and click Up. Alabama

is now in the first position, and Arizona in the second. You could have selected any item and used the Up and Down buttons to change its position in the string list.

The result is a string list whose `#define` name is `"SL_USA"`, and whose contents are the three strings `"Alabama"`, `"Arizona"`, and `"Georgia"`. To retrieve the entire string list into your application program, you should call `XVT_GlobalAPI::ListResStrings`, whose function prototype is the following:

```
void
ListResStrings(
    XVT_StrList*  dest,
    long          rid_first,
    long          rid_last)
```

The second and third arguments to `ListResStrings` are the `#define` names for the list (`"SL_USA"`) with `"_FIRST"` and `"_LAST"` appended, respectively. Your call to `ListResStrings` would therefore look something like this:

```
XVT_StrList* x;
ListResStrings( *x, SL_USA_FIRST, SL_USA_LAST );
```

Following the call to `ListResStrings`, the contents of the `XVT_StrList x` will be the strings you created using `XVT-Design++`.

**See Also:** To learn more about accessing and manipulating the `XVT_StrList` using the `XVT_GlobalAPI` member functions, see the *XVT++ 2.0 Class Library Reference*.

## 4.9. Userdata Strings

The Userdata feature in `XVT-Design++` lets you associate arbitrary data (up to six text strings) with any control, dialog box, window, or menu you create. For instance, you could associate textual data with a dialog box control.

**Example:** Imagine an application in which a user could perform various queries on a separate database program by pressing buttons in a dialog box. With the Userdata feature in `XVT-Design++`, you could associate the query commands needed for the database with a particular control in the dialog box. Then you could write C++ code that would send the associated userdata strings (the query commands) to the database, whenever the control was selected.

Userdata is stored in the application's URL file. As a result, you can change the userdata for an application without recompiling the application.

XVT-Design++ lets you create and edit userdata to be associated with an interface object. You can also change the labels that are associated with the userdata items themselves.

The XVT++ API provides three functions for accessing the userdata you have created with XVT-Design++:

- XVT\_GlobalAPI:GetWindowUserData
- XVT\_GlobalAPI:GetMenuUserData
- XVT\_GlobalAPI:GetDialogUserData

### 4.9.1. Creating Userdata

▼ To create userdata for a window, dialog box, control, or menu:

1. Select the desired interface object.
2. From the Edit menu, choose Userdata.  
The Edit Userdata window is displayed, as shown in Figure 50.

In the Edit Userdata window, you can create or modify six different userdata strings. These are the userdata strings that will be associated with the interface object you have created.

**Note:** Before using the Userdata command on a window or dialog box, make sure that no controls are currently selected within the window or dialog box.

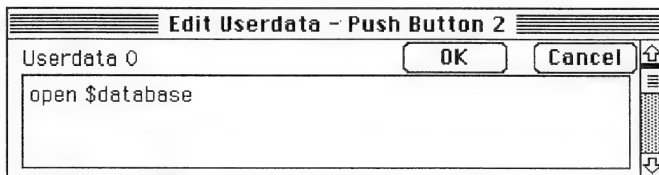


Figure 50. The Edit Userdata Window

### 4.9.2. Editing Userdata

The interface for editing userdata differs, depending on your platform:

- Motif systems: The userdata window presents all six userdata strings at once. Each of these strings is a separate text edit object. To move to a different text edit object, click on it.
- Non-Motif systems: The userdata window displays only a single text edit object. To move between text edit objects, use

the vertical scrollbar in the window. Performing a line-down or page-down operation on the scrollbar displays the next userdata text edit object; line-up or page-up displays the previous userdata text edit object.

Each text edit object can display three lines of userdata, which can include newlines. Newlines show up as “\n” in the URL file. You can enter an unlimited number of lines of userdata into each userdata string.

### 4.9.3. Userdata Labels

In the Edit Userdata window, userdata labels are associated with userdata strings. Userdata labels are simply titles or descriptions of each of the six userdata strings that can be associated with interface objects. The default userdata labels are “Userdata 0”, “Userdata 1”, ... “Userdata 5”.

Userdata labels appear only in the Edit Userdata window of XVT-Design++. They are not placed in the generated source code or URL files. The labels are just to remind you what a particular userdata string means.

Because userdata labels are on a project-wide basis, you can create labels before you have created a window, dialog box, menu, or control. You can also change the userdata labels even if you have not created any of these interface objects. You can edit userdata labels without having to edit the userdata itself.

▼ To edit userdata labels:

1. From the Edit menu, choose Userdata Labels.  
The Edit Userdata Labels dialog box appears as shown in Figure 51.

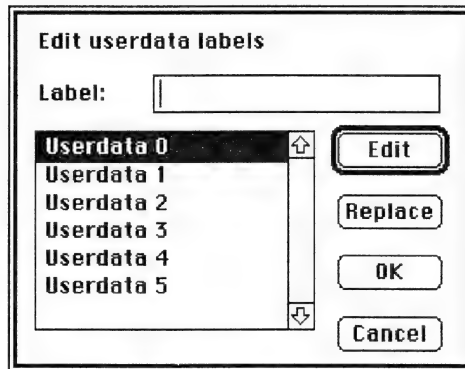


Figure 51. The Edit Userdata Labels Dialog

2. In the Edit Userdata Labels dialog, select a label in the list box.
3. Click Edit (or double-click the label).  
The label appears in the edit field.
4. Edit the label, then use the controls to change the label.

#### **Edit**

Puts the selected label into the edit field.

#### **Replace**

Replaces the existing label with your new label.

#### **OK**

Accepts the changes that you have made to the labels and removes the dialog box. After you click OK, any future Edit Userdata window that you bring up displays the new userdata labels above each of the six userdata strings.

#### **Cancel**

Removes the dialog box without applying the changes you have made to the userdata labels.

#### 4.9.4. Generating Code with Userdata

When you generate an application, the userdata is written to the project URL file as a `userdata` statement in a dialog, window, or menu definition. If the userdata is associated with a control, the `userdata` URL statement follows the control definition in the URL file.

### 4.10. TestMode

In XVT-Design++, TestMode lets you verify the appearance of your application's user interface without compiling or linking. TestMode emulates your application's behavior at runtime, as if you had compiled, linked, and executed it.

During TestMode, the XVT-Design++ user interface is replaced with your application's. Instead of seeing the XVT-Design++ windows and dialogs, you'll see those of your project.

In TestMode, your project's menus, windows, dialogs, and controls behave as if your application were actually running. If you induce any events for which you have defined connections (for example, choosing menu items or manipulating controls), the actions for those connections are executed. You can open windows by choosing menu items, create or destroy dialogs by clicking push button controls, and so forth.

If you use XVT-Design++ on more than one platform, you can use TestMode to check your application's appearance on each platform. Project files from XVT-Design++ are portable across all supported platforms.

**Note:** Keep in mind that XVT-Design++ *emulates* the behavior of your application. It does *not* interpret or execute your program's code. TestMode ignores any code you have entered with the ACE. You do not need to generate any source files before using TestMode. The presence or absence of generated files has no effect.

#### 4.10.1. Entering TestMode

▼ To test your project:

1. From the Tools menu, choose Enter TestMode.  
XVT-Design++ first asks if it should save changes to the project (only when there are unsaved changes). Next, XVT-Design++ hides its menubar and any open windows and dialogs. The Task Window and its menubar are replaced with your project's Task

Window and menubar. A special “TestMode” menu is appended to the right of your application’s menubar.

2. Test any menus, windows, dialogs, and controls in your project for which you have defined connections.

**Note:** The reason that XVT-Design++ asks if it should save changes when entering TestMode is that it is possible to trap yourself within TestMode. For example, you could create a connection to open a modal dialog that has no way to close it.

When you enter TestMode, your application receives a Create event. If you want a window or dialog to appear as soon as your application starts, create a connection for the application’s Create event tag to create the window or dialog.

The special TestMode menu appears on all of your application’s menubars while you use TestMode, to provide a way for you to exit TestMode. It does not appear in your compiled, standalone application.

**Note:** If you haven’t created any connections in your project, TestMode is not very interesting. You will see only your application’s Task Window and its menubar. Choosing menu items won’t have any effect, except choosing Quit from the File menu, which will cause XVT-Design++ to leave TestMode.

#### 4.10.2. Leaving TestMode

▼ To leave TestMode:

From the TestMode menu in your application’s Task Window, choose Exit Test.

-OR-

From your application’s File menu, choose Quit.

-OR-

Close your application’s Task Window, if the native window system provides a way to do this.

**Note:** If you have defined a connection for the Quit item on a File menu, choosing this item will *not* terminate TestMode.

When you leave TestMode, XVT-Design++ redraws its menubar and reopens any windows that were open before you entered TestMode. The last event that your application receives before it is closed and XVT-Design++ returns from TestMode is Destroy.

### 4.10.3. Special Considerations for TestMode

Although XVT-Design++ emulates your application's runtime appearance and behavior as accurately as possible, there are a few limitations.

#### 4.10.3.1. About Box

In TestMode, you will not see your application's About dialog box. A dummy dialog box is displayed in its place.

#### 4.10.3.2. Help File

In TestMode, you cannot examine your application's help file. If you invoke on-line help in TestMode, you will see a help file describing TestMode, rather than your help file.

## 4.11. File Generation

Once you've created the windows, dialog boxes, menus, and other resources for your project with XVT-Design++, the next step is to generate the source code for your application. XVT-Design++ automatically generates Universal Resource Language (URL) files for your resources, and C++ source code files to handle these resources in your final application.

### 4.11.1. Destination Directory

By default, XVT-Design++ puts the files it generates in the same directory as the project file.

▼ To change the destination directory:

1. From the File menu, choose Generated Files.  
The Generated Files dialog appears, as shown in Figure 52. The current destination directory is displayed at the top of the dialog.

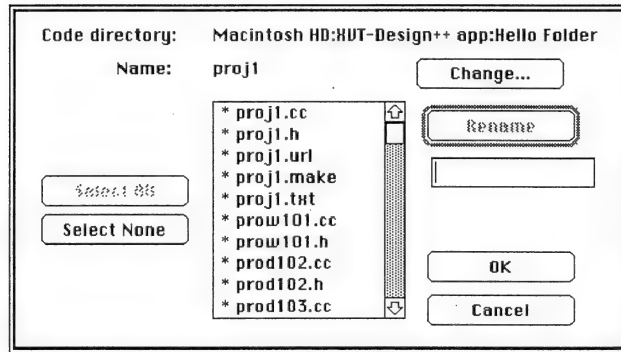


Figure 52. The Generated Files Dialog

2. Click the Change button. A standard save-file dialog appears.
3. Navigate to the directory in which you want XVT-Design++ to place your application's file.
4. Click the Save button.

**Note:** This procedure does not actually create any files; it simply sets the destination directory for when the files are generated.

#### 4.11.1.1. The Application Name

The name of your application is distinct from the name of its project file. For instance, the name of your project file might be "MDB30.PRJ", but the application it generates could be named "Mega Database 3.0".

You set the name of your application when you set the destination directory, as described in the previous section. Before clicking the Save button in the save-file dialog, type the name of your application in the filename edit control in that dialog.

#### 4.11.2. Filenames

XVT-Design++ creates names for the files it generates, based on several pieces of information:

- The name of the application (which you supply)
- The type of the file
- The resource associated with the file

In most cases, the first three characters of each file's name will be the same as those of the application's name.

In the filenames given in the following sections, we indicate these first three characters as “<xxx>”, and the complete application name as “<appname>”. For example, if your application is named “hello”, the filename described as “<xxx>d101.cc” would actually be “held101.cc”.

You can change these names as you desire. See section 4.11.4.

### 4.11.3. Types of Generated Files

XVT-Design++ generates several different kinds of files for your application: C++ source code files, a header file, a resource file, a help text file, and a makefile. The following sections describe these files and their naming conventions.

#### 4.11.3.1. C++ Source Code Files

XVT-Design++ creates a separate .cc file for each module in your project (including the application itself). These files contain subclasses and event handler member functions for the module, as created with the Action Code Editor.

The names for these files have the following format:

**<xxx><r><nnn>.cc**

where “<xxx>” is the first three characters of the application's name. “<r>” is a character indicating the type of the associated resource, which is interpreted as follows:

**d** - dialog box

**w** - window

“<nnn>” is the ID number of the resource (three digits). Resources are numbered consecutively, in order of their creation.

The main function and the Task Window's event handler member functions are placed in a file called **<appname>.cc**.

**Example:** Suppose your application's name is “Hello”, and it has one window and one dialog. The source code file for the window will be named **helw101.cc**, and the file for the dialog will be named **held102.cc**.

#### 4.11.3.2. Header File

XVT-Design++ creates one header file for the application, named **<appname>.h**. It contains the Task Window class and resource ID definitions. A header file is also created for each derived container class, named **<xxx><r><nnn>.h**. These contain class definitions for the interface objects.

#### 4.11.3.3. Resource File

XVT-Design++ generates one XVT Universal Resource Language (URL) file, named **<appname>.url**. It contains resource descriptions for all of the modules in your application.

#### 4.11.3.4. Help Text File

XVT-Design++ creates a file containing the text for your application's on-line help. It is named **<appname>.txt**.

#### 4.11.3.5. Makefile

XVT-Design++ creates a complete makefile for compiling and linking your application's source files. It is named **makefile.<xxx>**.

### 4.11.4. Choosing Files to Generate

The first time you generate source files for a given project, you will probably want to generate all of these files. Later, when you modify the project (for instance, to change the items in a menu, or add a new dialog box) you may not need to regenerate all the files.

▼ To choose which files to generate:

1. From the File menu, choose Generated Files.  
The list box in the dialog lists all the files that are part of your application, and whether or not they are to be generated. If an asterisk ("\*") precedes the filename, XVT-Design++ will generate a new copy of that file. If no asterisk precedes the filename, XVT-Design++ will not generate a new copy.
2. To turn a file's asterisk off or on, double-click the filename.
3. To enable generation of all the files, choose Select All. To disable generation of all the files, choose Select None.

▼ To change the name of a generated file:

1. In the list box, click on the file's name.  
The name appears in the edit control on the right.
2. In the edit control, change the file's name.
3. Click the Rename button.

**Caution:** XVT-Design++ overwrites existing files without warning. If any files in the destination directory have the same name as files listed in the Change Filenames dialog, they will be overwritten when XVT-Design++ generates the files. Hence you should not change the contents of generated files outside of XVT-Design++. Instead, always use the Action Code Editor to add code to your project.

## 4.11.5. Makefiles

XVT-Design++ can generate makefiles that work with over 20 different combinations of platforms and compilers. It uses pre-defined "templates" for each different configuration, and adds information about the particular files for your project when it generates the makefile.

By default, XVT-Design++ uses a template that is appropriate for the platform you are running on. However, you can choose to generate a makefile for any of the available configurations.

### 4.11.5.1. Template-based Generation

Whenever you generate your application and you have included a makefile among the files to generate, XVT-Design++ writes the selected template. By default, it uses a template appropriate for the platform you are running on. From the template, XVT-Design++ generates a working makefile.

**Example:** If you are running XVT-Design++ on the Macintosh, then by default XVT-Design++ generates a makefile that will build your XVT++ application using Design++/Mac and the MPW C++ compiler.

To see this, select Configure Makefile from the File menu. The dialog shown in Figure 53 appears.

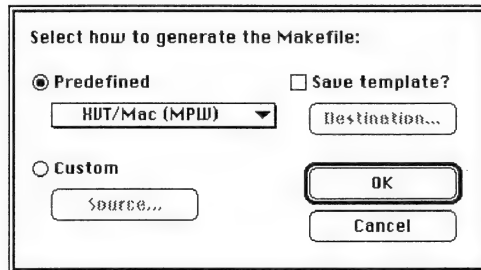


Figure 53. The Configure Makefile Dialog

Notice that the “XVT/Mac (MPW)” item is selected in the “Predefined” template list button, since this is the default for the Design++/Mac system. To generate a working makefile for any of the other supported configurations, simply choose the appropriate one from the list button, and click OK.

#### 4.11.5.2. Custom Templates

The template-driven makefile system in XVT-Design++ does more than simply provide working makefiles for dozens of different systems. It also allows you to customize the generated makefile for your particular installation.

▼ To create a custom template:

1. In the Configure Makefile dialog, select Save Template.
2. To choose a filename for the template, click the Destination push button.  
A copy of the temporary template file is placed on disk for you to read and possibly modify.

For example, you might want to modify the compile or link options, or change the include directories for finding header files. This can be accomplished by changing the template file on disk.

▼ To use the custom makefile:

1. In the Configure Makefile dialog, choose Custom.
2. To tell XVT-Design++ to use your customized template file, click the Source button.  
XVT-Design++ reads your modified makefile template when it generates the working makefile. The makefile now includes your modifications.

#### 4.11.5.3. The XVTDIR Macro

Before generating a makefile, you must make sure that the XVTDIR macro in your **designpp.cfg** configuration file has been set to the appropriate location for the target destination for which you are generating code.

**See Also:** For more information on the XVT-Design++ configuration file, see section 5.4.

#### 4.11.5.4. External Files

Source-code files that are part of your application's code, but not generated by XVT-Design++, are called external files. External files are usually used for portions of your program other than the user-interface code.

You can add external file references to your XVT-Design++ project. Then, when XVT-Design++ generates the makefile for your application, it inserts dependency information for your external files.

▼ To add external file references:

1. From the File menu, choose External Files. The External Files dialog appears.
2. Type the names of your external files in the dialog. Place each filename on a separate line. Each file can depend on one or more other files. Dependencies are described as follows:

`<dependent_filename>: <filename1> <filename2> ...`

**Note:** When generated, the makefiles automatically convert these dependencies to whatever form is required by make on a particular platform.

**Example:** In Figure 54, **mycode.cc** and **dsp.cc** are external files: **mycode.cc** includes **mycode.h**, and **dsp.cc** includes **dsp.h**, **stdio.h**, and **math.h**.

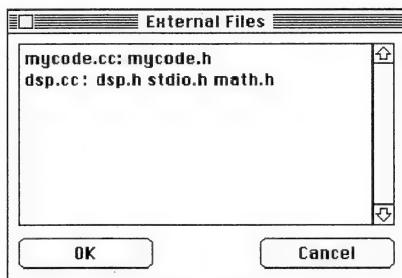


Figure 54. Dependent External Files

# 5

---

## REFERENCE

### 5.1. Menu Commands

This section contains brief descriptions of all the commands on the XVT-Design++ menus. For more detailed descriptions of the dialogs and operations described in this section, see Chapter 4.

#### 5.1.1. File Menu

The File menu contains commands for manipulating project files, and for generating your application's source code and resource files.

##### **New Project**

Creates a new project. This project becomes the active one in XVT-Design++. If other projects are open when you choose this command, XVT-Design++ hides their windows. See section 4.1.

##### **Open Project**

Presents a standard open-file dialog, so you can open a previously saved project file. When you open a project, it becomes the active XVT-Design++ project. If other projects are open when you choose this command, XVT-Design++ hides their windows. See section 4.1.

**Note:** The list of opened projects is appended to the Edit menu to facilitate switching between them.

### **Close Project**

Closes the active project. If you have made any changes to the project since it was last saved, XVT-Design++ asks whether you want to save the project before it is closed. See section 4.1.

### **Save Project**

Saves the active project file. The first time you use this command for a new project, a standard save-file dialog appears. You can choose the directory in which to save the file, and enter the project's name. See section 4.1.

### **Save Project As**

Opens a standard save-file dialog, allowing you to choose a new filename and directory for the active project. See section 4.1.

### **Generate Application**

Creates the resource file, the makefile, and all the source code files for your application. The files are saved in the destination directory, as specified in the Filenames dialog. (To open the Filenames dialog, choose "Generated Files" from the File menu.) See section 4.11.

### **Generated Files**

Opens the Filenames dialog, where you can change the names of the source code files and resource files for your application. You can also change the directory in which XVT-Design++ places your application's files. See section 4.11.

### **Configure Makefile**

Opens the Makefile dialog, where you can choose which predefined makefile template will be used to generate your application's makefile. Or, you can choose a custom makefile stored in an external file. See section 4.11.

### **Quit (Exit on MS-Windows)**

Closes all open projects and terminates execution of XVT-Design++. If any open projects have been changed since they were last saved, XVT-Design++ asks if you want to save them before closing.

### **5.1.2. Edit Menu**

The Edit menu contains commands for editing text and interface objects, and for invoking the various editor windows in XVT-Design++.

#### **Cut**

Removes the selected text or interface object and places it on the system clipboard.

#### **Copy**

Places a copy of the selected text or interface object on the system clipboard.

#### **Paste**

Copies the contents of the system clipboard to the Layout or ACE window that has focus.

#### **Clear**

Permanently removes the selected text or interface object without placing it on the clipboard.

#### **Select All**

Selects all text or interface objects in the Layout window with focus.

#### **Userdata Labels**

Opens the dialog for editing userdata label strings. See section 4.9.3.

#### **Project Attributes**

Opens the dialog for setting the project attributes: the About dialog, the Task Window's menubar, the resource base name, and the Task Window title. See section 4.1.

#### **Attributes**

Opens the attributes dialog for the selected interface object, or the layout window that has focus. See section 4.5.

### **Code**

Opens an Action Code Editor window for the selected interface object, or the layout window that has focus. See section 4.2.

### **Userdata**

Opens a userdata editor window for the selected interface object, or the layout window that has focus. See section 4.9.

### **Menu**

Invokes the Menu Editor for the menubar associated with a window. If no menu is associated with a window, or if no window is currently active, this menu item is disabled. See section 4.7.

### **Creation Order**

Opens the Creation Order dialog, which lets you set the order in which controls in a dialog receive focus during keyboard navigation. See section 4.6.

**Note:** The list of opened projects is appended to the Edit menu to facilitate switching between them.

## **5.1.3. Tools Menu**

The Tools menu contains commands for opening the various editor dialogs in XVT-Design++, and for starting TestMode.

### **Action Code Editor**

Opens an Action Code Editor dialog. If a layout window has focus, the Action Code Editor's context list buttons will be set to the container, or the selected control. Otherwise, the context list buttons will be set to the application. See section 4.2.

### **Menubar Editor**

Invokes the Menubar Editor, so you can create and modify menubars and menus. If the Menubar Editor's dialog is already open, choosing this command brings it to the front. See section 4.7.

### **Strings Editor**

Invokes the String and Stringlist Editor, so you can create and modify string resources. See section 4.8.

### **Begin TestMode**

Starts TestMode in XVT-Design++, so you can examine and demonstrate your application's resources as they will appear at runtime. To leave TestMode, choose End TestMode from the TestMode menu or choose the application termination option from your menus. See section 4.10.

## **5.1.4. Window Menu**

The Window menu includes commands for creating new windows and dialogs. When you create a new container, XVT-Design++ opens a layout window for it. The Window menu also lists all the currently open windows. This menu is updated as you open and close layout windows. Choosing an item from the menu brings the corresponding window to the front. On the menu, any open Action Code Editor windows are listed before the layout windows. See section 4.3.

### **New Window**

Creates a new window resource in the active project. See section 4.3.

### **New Dialog**

Creates a new dialog resource in the active project. See section 4.3.

## **5.1.5. Controls Menu**

The Controls menu includes commands for creating the various types of controls available in windows and dialog boxes: push buttons, check boxes, radio buttons, scrollbars, static text, edit controls, text edit objects, list boxes, list buttons, list edits, group boxes, and custom controls.

### **▼ To create a control:**

1. From the Controls menu, choose the control type.
2. Position the cursor in the upper left corner of the desired location.
3. Either click or drag the control into the desired size.  
If you click to create the control, it will be of the standard size for this type of control.

▼ To create multiple controls of the same type:

1. Press the Shift key while pulling down the Controls menu to choose a control.  
The cursor becomes a “+” to indicate visually that you are in multiple-control creation mode.
2. Click or drag repeatedly to create several copies of the control.
3. To exit from this mode, choose Pointer (or another control) from the Controls menu.

When the Pointer tool is selected, you can select, resize, or move existing controls. The Pointer is also used to return to pointer mode after you have gone into multiple-control creation mode.

### 5.1.6. Layout Menu

The Layout menu contains commands for adjusting the position of controls in window and dialog layout windows. This menu is available only in layout windows. If no controls are selected, the commands are disabled.

The order in which you select controls determines how the alignment commands will affect the controls. The first control selected is the reference point used to align the other controls.

**Example:** For example, if you select controls A, B, and C (in that order), and then choose Align Left from the Layout menu, controls B and C will be lined up along the left boundary of control A, because A was the first one selected.

**Note:** If you use Select All from the Edit menu or select a set of controls by dragging out a boundary with the mouse, the reference control used to align the other controls is the earliest of the selected controls in terms of creation order.

#### Align Left

Aligns the selected controls along their left boundary. See section 4.4.1.

#### Align Center

Aligns the selected controls along their horizontal center without changing their vertical positions. See section 4.4.1.

### **Align Right**

Aligns the selected controls along their right boundary. See section 4.4.1.

### **Align Top**

Aligns the selected controls along their top boundary. See section 4.4.1.

### **Align Middle**

Aligns the selected controls along their vertical center without changing their vertical positions. See section 4.4.1.

### **Align Bottom**

Aligns the selected controls along their bottom boundary. See section 4.4.1.

### **Even Horizontal Spacing**

Adjusts the horizontal position of the selected controls so that they are separated horizontally by the same distance. See section 4.4.2.

### **Even Vertical Spacing**

Adjusts the vertical position of the selected controls so that they are separated vertically by the same distance. See section 4.4.2.

### **Make Same Size**

Makes all selected controls the same size as the control first selected. See section 4.4.2.

### **Grid**

Opens the Grid Settings dialog, where you can set the grid spacing interval, and choose whether the grid is displayed. See section 4.4.3.

## **5.1.7. Font and Style Menus**

The Font and Style menus let you set the font, font size, and style currently used for text in the Action Code Editor. You can use different fonts for different ACE windows. If more than one ACE window is open, choosing items from the Font and Style menus affects the front-most window.

If you want to permanently change the default font for the ACE window, modify **designpp.cfg** as described in section 5.4.3.5.

**Note:** On some systems (for example, MS-Windows) the Font and Style menus are combined into a single Font menu.

## 5.2. Interface Objects and Tags

This section describes all the tags available for the standard XVT Portability Toolkit user interface objects. To see which tags are available for each interface object, refer to the table at the end of this section.

### 5.2.1. Tag Descriptions

This section describes the tags available in XVT-Design++. There are two types of tags:

- **Event tags** correspond to the events that are generated by user actions (for example, mouse clicks) or by the window system (for example, update events)
- **Special tags** do not correspond to events but rather serve as convenient places to add items like member functions and state variables to the generated code

#### 5.2.1.1. Event Tags

The following tags correspond directly to XVT++ `e_*` event handler member functions. In the Action Code Editor's Tag list button, event tags appear with an "EVNT:" prefix.

**See Also:** For more detailed descriptions of these events, refer to the *Portability Toolkit Programmer's Reference*.

##### Action

The action associated with selecting a control or menu item should be taken.

##### Char

The user pressed a key on the keyboard.

##### Close

The user operated the close control or close menu item on a window or dialog.

##### Create

A window or dialog has been created. This is the first event that a window or dialog receives; the first event sent to the application is a Create event for the Task Window.

**Destroy**

A window or dialog has been closed. This is the last event received by a window or dialog.

**Focus**

A window or dialog has gained or lost keyboard focus.

**Font**

The user has chosen an item from the standard Font menu.

**HScroll**

The user has manipulated the horizontal scrollbar on the frame of a document window.

**Mouse\_Dbl**

The mouse button was double-clicked in a window.

**Mouse\_Down**

The mouse button was pressed in a window.

**Mouse\_Move**

The mouse pointer was moved in a window.

**Mouse\_Up**

The mouse button was released in a window.

**Quit**

The window system is shutting down (not available on all platforms).

**Size**

The size of a window or dialog has been set or changed.

**Timer**

The timer associated with a window or dialog has gone off.

**Update**

The contents of a window require redrawing.

**User**

An application-defined event has been initiated.

**VScroll**

The user has manipulated the vertical scrollbar on the frame of a document window.

### 5.2.1.2. Special Tags

The following tags mark convenient places in the generated code for adding items such as instance data, class definitions, or member functions. In the Action Code Editor's Tag list button, special tags appear with an "SPCL:" prefix.

**Bottom**

The place to add member function declarations for windows and dialogs.

**Class Decl**

The place to add member function declarations or instance data to the class definition for an interface object.

**Class Header**

The place to add `#include`'s to header files for dialogs and windows.

**Constructor**

The code that will be placed in the constructor for the object.

**Destructor**

The code that will be placed in the destructor for the object.

**Help**

The help text associated with an interface object. Text for this tag is placed in your application's help-text file. (By default, this file has a `.txt` extension.)

**Is Quit OK**

This tag lets you define a `QuitOK` member function for the generated class derived from `XVT_TaskWin`. This function is called from within the `e_quit` member function of the Task Window. The function's return value indicates if the application is ready to terminate.

**Main Code**

Text for this tag is inserted in the `main` function of your application, immediately before calling the `Init` member function for the generated class derived from `XVT_TaskWin`. It is the first application code that is executed, up through the call to the `Init` member function for the application class.

**Top**

The place to add declarations, constants, or includes needed to compile this object.

**User Header**

Text for this tag is placed at the beginning of the header file for your application. Use this tag to declare application-wide constants or class definitions. Also use it to include header files needed by the entire application. The definitions in this tag will be available to all other tags.

**User URL**

Text for this tag is inserted in your application's URL file. The text is placed before the resource statements for your

application's resources. It marks the place to add native resource definitions or to include resource definitions unknown to this XVT-Design++ project.

## 5.2.2. Interface Object/Tag Pairs

Tables 1 and 2 show which event and special tags, respectively, are available for each type of user-interface object. User-interface objects available within XVT-Design++ and associated with tags include the following:

- Task Window
- Screen windows
- Menu items
- Dialogs
- Controls

Controls include:

- Push buttons
- Check boxes
- Edit fields
- Group boxes
- Static text
- List boxes
- List buttons
- List edits
- Radio buttons
- Scrollbars (both horizontal and vertical)

**Note:** No tags are associated with menubars and text edits.

	Task Window	Screen Window	Menu Item	Dialog	Controls
Action			•		•
Char		•		•	
Close	•	•		•	
Create/Destroy	•	•		•	•
Focus		•		•	
Font	•	•			
Hscroll/Vscroll		•			
Mouse*		•			
Quit	•				
Size	•	•		•	
Timer	•	•		•	
Update		•			
User	•	•		•	•

Table 1. Interface Object/Event Tag Pairs

	Task Window	Screen Window	Menu Item	Dialog	Controls
Help	•				
Is Quit OK	•				
Main Code	•				
User Header	•				
User URL	•				
Class Decl	•	•	•	•	•
Class Header		•		•	
Top	•	•		•	
Bottom	•	•		•	
Constructor	•	•	•	•	•
Destructor	•	•	•	•	•

Table 2. Interface Object/Special Tag Pairs

### 5.3. Variables and Constants in Action Code

The variables, constants, and functions available at any point are defined by the class structure and hierarchy in XVT++.

One special local variable, long `xdReturnValue`, is available and initialized to 0L in User tag code for the interface objects that receive it: Task Window, screen window, dialogs, and controls. In cases where the superclass of the object is not the default XVT++ class, `xdReturnValue` receives the result from the `e_user` member function of the superclass.

The protected data member `xdContainer` is generated in all `MenuItem` and `Control` subclasses, and is available to any member function.

### 5.4. The Configuration File

XVT-Design++ uses a configuration file to determine default settings for many user-changeable options. You can edit the configuration file to change these default values, so they suit the

characteristics of your development environment and your preferences.

XVT-Design++ examines the contents of the configuration file only when it starts up. Any changes you make to the file while XVT-Design++ is running have no effect.

**Note:** The configuration file is not mandatory. XVT-Design++ still runs normally even if it does not locate the file.

### 5.4.1. Name and Location

The configuration file is named **designpp.cfg**. The configuration file can be placed in the same directory as the XVT-Design++ application or in another directory, depending on the operating system. This section describes these directories, by operating system.

#### 5.4.1.1. Macintosh

On the Macintosh, when XVT-Design++ is launched, it looks for its configuration file in the folder that contains the application itself.

#### 5.4.1.2. OS/2

Under OS/2, when XVT-Design++ is launched, it first looks for its configuration file in the current directory. If it does not find the configuration file there, it looks in the root directory of the volume that contains XVT-Design++.

#### 5.4.1.3. UNIX

Under UNIX, when XVT-Design++ is launched, it first looks for its configuration file in the current directory. If it does not find the configuration file there, it looks in the user's **HOME** directory.

#### 5.4.1.4. Windows

Under Windows, when XVT-Design++ is launched, it first looks for its configuration file in the current directory. If it does not find the configuration file there, it looks in the root directory of the volume that contains XVT-Design++.

### 5.4.2. Format

The configuration file is a plain text file. You can use any text editor to create, examine, or change it.

### 5.4.2.1. Commands

Commands in the configuration file are placed on separate lines, one entry per line. Entries must be in the following format:

```
<attribute>: <value>
```

where <attribute> is one of the attribute names listed in Section 5.4.3, and <value> is an integer, string, or Boolean constant. Booleans are denoted as follows:

```
FALSE:false, f, no, off, 0
```

```
TRUE:(anything not defined as FALSE)
```

Uppercase and lowercase are not distinguished in Boolean values.

### 5.4.2.2. Comments

You can add comments to the configuration file by placing a pound sign, #, at the beginning of the line. All subsequent information on the line is ignored. For example:

```
# This is a comment.
```

## 5.4.3. Available Options

This section lists all the attributes that can be specified in the configuration file, organized by category. The required value type appears in parentheses after the attribute name.

### 5.4.3.1. Default Grid Settings

These attributes set the default values for the alignment grids in layout windows. When you create a new window or dialog, its layout window initially has these grid settings. You can change these settings in individual layout windows by using the Grid command on the Layout menu.

**gridDisplay** (Boolean)

If TRUE, the alignment grid is visible in layout windows.

**gridNative** (Boolean)

If TRUE, the alignment grid's spacing interval is based on the average height and width of a character of the system font.

**gridSnap** (Boolean)

If TRUE, controls automatically align to the grid when placed, moved, and resized in layout windows.

**gridX, gridY** (integer)

These attributes set the spacing interval for the alignment grid, in pixels.

### 5.4.3.2. Makefile Template Macros

These options let you create string-substitution macros for your makefile templates.

**userMacros** (one or more strings, separated by commas)

Use this option to declare the names of your macros. You must declare each macro before defining it.

**<macro name>\_definition** (string)

Use this option to define a macro. The string value of this option will be substituted for each appearance of <macro\_name> in your makefile template, when your application's makefile is generated.

**Example:** The following is an example of using the macro options. These lines would appear in your **designpp.cfg** file:

```
userMacros:string1, dude1, dude2, XVTDIR
XVTDIR_definition: c:\xvt\win31
string1_definition: Party on
dude1_definition: Wayne
dude2_definition: Garth
```

These lines would appear in your makefile template:

```
%{string1}, %{dude1}! %{string1}, %{dude2}!
XVTDIR = %{XVTDIR}
```

When your application's makefile is generated, the two previous lines would be expanded to:

```
Party on, Wayne! Party on, Garth!
XVTDIR = c:\xvt\win31
```

### 5.4.3.3. Custom Template Names

These options let you add your makefile template files to the list of templates in XVT-Design++.

**userTemplates** (string)

This option declares the symbolic names for your makefile template files. These names are used only within the configuration file. To specify more than one file, separate their names with commas.

**<template\_name>.title** (string)

This option defines the title of your template. The title appears in the list button in the XVT-Design++ Configure Makefile dialog. If a predefined template has the same title, the custom template file replaces the predefined template.

**<template\_name>.filename** (string)

This option defines the filename of your template. Because XVT-Design++ uses this name to open your template, it must be a valid filename for your development platform.

**Example:** The following is an example of using the custom template options. These lines would appear in your **designnpp.cfg** file:

```
userTemplates: one, two
one.title: Doug's New Template
one.filename: doug.tpl
two.title: Design++/Win (MSC)
two.filename: new_win.tpl
```

This example adds a new template called “Doug’s New Template” to the list of templates in the Configure Makefile dialog, and replaces the predefined template for the Microsoft C++ compiler. The templates’ filenames are **doug.tpl** and **new\_win.tpl**, respectively.

#### 5.4.3.4. File Defaults

**macCreatorCode** (four-character string)

This attribute sets the creator code for all source files generated by XVT-Design++. This attribute is useful only on the Macintosh version of XVT-Design++.

**macFileType** (four-character string)

This attribute sets the file type code for all source files generated by XVT-Design++. This attribute is useful only on the Macintosh version of XVT-Design++.

**projectFileExtension** (string)

This attribute sets the suffix string for project filenames. This suffix has different effects on different platforms. For example, it is restricted to three characters on Windows.

#### 5.4.3.5. ACE Text Defaults

These options set text characteristics for the Action Code Editor’s text-editing pane.

**aceFontFamily** (string)

This option sets the default font. The value should be one of the following:

**FF\_FIXED**: a fixed-width (monospaced) font, such as Courier

**FF\_SANS\_SERIF**: a plain, sans-serif font, such as Helvetica

**FF\_SERIF**: a serif font, such as Times

The actual font used depends on your platform. If no default font is specified, the system font is used.

**aceFontSize** (integer)

This option sets the default font size, in points. The default value is 12.

**tabStop** (integer)

This option sets the distance between tab stops, in characters. The default value is eight.

#### 5.4.3.6. Miscellaneous Options

**cfgLogfile** (string)

This option lets you explore how XVT-Design++ uses the configuration file. If this attribute is set to a filename that can be opened for writing with `fopen`, XVT-Design++ creates a text file that lists all inquiries made concerning user-configurable options:

- If an inquiry is made for an option that is specified in the configuration file, an entry of the form “<attribute>:<value>” is added to the log file.
- If the option is not specified in the configuration file, an entry of the form “#<attribute>:” is added. Both of these entries are in the same format as entries for the configuration file itself. You can add options to your configuration file by copying them from the log file.

**Note:** The log file may contain redundant entries, since XVT-Design++ makes an entry for each query to a given option.

**initialAboutBox** (Boolean)

If this attribute is `FALSE`, XVT-Design++ does not display its version and copyright dialog box when starting up.

#### 5.4.4. Configuration File Example

Here is an example of a configuration file:

```
#Example designpp.cfg file
#Turn off the splash screen:
initialAboutBox: false
#Set a 16x16 pixel alignment grid:
gridX: 16
gridY: 16
gridNative: false
#Change the project file extension so as to
#not conflict with Borland's project files:
projectFileExtension: xdp
```

# XVT-DESIGN++

## INDEX

### A

About box, 32, 33, 40  
     in TestMode, 91  
 abstract behavior class, 11  
 Accelerator, 81  
 ACE, see Action Code Editor  
 aceFontFamily, 115  
 aceFontSize, 116  
 action code, 15, 37, 42  
 Action Code Editor, 4, 14, 19, 54, 102, 115  
     controls, 55  
     invoking, 54  
 Action tag, 106  
 Align Bottom, 63, 105  
 Align Center, 62, 104  
 Align Left, 62, 104  
 Align Middle, 63, 105  
 Align Right, 62, 105  
 Align Top, 62, 105  
 aligning controls, 61, 104  
 Alt/Control/Shift, 81  
 applications  
     attributes, 33  
     building, 17, 52  
     generating, 100  
     Hello, 18  
     implementing behavior, 11  
     sample, 17  
     setting name for, 51, 92  
 Attributes, 59, 78, 101

attributes, 27  
     application, 33  
     dialogs, 30, 71  
     interface object, 13  
     resource identifier, 13  
     title, 13  
     windows, 72  
 Autohscroll, 69  
 Autovscroll, 69

### B

Before/After, 79  
 Begin TestMode, 103  
 behavior classes, abstract, 11  
 behavior object, 11  
 binary files, 53  
 Border, 70, 72  
 Borland C++ IDE, 52  
 Bottom tag, 108  
 building applications, 17, 52

### C

C++  
     classes, 7  
     source code files, 93  
 C++ programming, 7  
     prerequisites, 2  
     reference books, 12  
     style, 11  
 Cancel control, 32, 67  
 .cc source code files, 65

- cfgLogfile, 116
- Change String, 58
- Char Limit, 70
- Char tag, 106
- character-based platforms, 64
- Checkable, 81
- Checked, 67, 81
- choosing files to generate, 94
- Class, 68, 71, 72
- Class Header tag, 108
- Class\_Decl tags, 42, 108
- classes
  - adding variables to, 43
  - C++, 7
  - container, 12
  - custom, 17
  - hierarchy, 8
  - names, 72
  - XVT++, 7
- Clear, 77, 78, 101
- Close Box, 73
- Close Object, 59
- Close Project, 100
- Close tag, 106
- Code, 89, 102
- code
  - action, 15, 37, 42
  - avoiding duplicate, 11
  - editing, 4, 14
  - generated, 14
  - generating with userdata, 89
  - source, 42, 50
  - user interface, 14
- configuration file, 111
  - commands, 113
  - comments, 113
  - format, 112
  - sample, 116
- Configure Makefile, 100
- connections
  - between objects, 4, 15
  - creating & editing, 57
  - removing, 59
  - setting, 34
  - testing, 40, 90
  - what happens when you create, 37
- constants
  - in action code, 111
- constructor member function, 12
- Constructor tag, 108
- Constructors, 42, 44, 46, 48
- container classes
  - GUI, 12
- containers, 26, 56
- context, in ACE, 15, 35, 55
  - setting, 35
- Control subclasses, 111
- controls, 56
  - aligning, 61
  - appearance of, 5
  - Cancel, 32, 67
  - creating, 60, 103
  - custom, 68
  - Default, 31, 67
  - in ACE, 55
  - multiple, 30, 61, 104
  - multiple similar, 11
  - push buttons, 28
  - radio buttons, 30
  - size & position, 13
  - sizing, 61, 66
  - spacing, 63
- Controls menu, 103
- Copy, 57, 70, 78, 101
- Copy command, 49
- Create Standard XVT Dialog, 58
- Create tag, 106
- creating
  - connections, 57
  - controls, 60, 103
  - dialogs, 29, 60
  - menubars, 19
  - multiple controls, 30
  - new menu, 21

- projects, 18, 54
- resources, 60
- windows, 27, 60
- Creation Order, 74, 102
- creation order
  - editing, 74
- custom classes, 17
- custom controls, 68
- customer support, 5
- Cut, 57, 70, 77, 101

## D

- Default Code, 59
- Default control, 31, 67
- #define, 81
  - name, 65
  - string, 13, 24
- delegate objects, 11
- designpp.cfg, 106, 112, 114, 115
- destination directory, changing, 91
- Destroy tag, 107
- destructor member function, 12
- Destructor tag, 108
- Destructors, 42
- development environments, 52
- dialogs
  - attributes, 30, 71
  - creating, 29, 60
  - modal, 71
  - modeless, 71
  - pre-defined, 38
  - size & position, 13
- Disabled, 67, 71, 73, 81
- Display Grid, 64
- Document Prefix string, 33
- Done, 77, 80
- duplicate code, avoiding, 11

## E

- e\_\* member functions, 14, 106
- e\_action, 12
- e\_create, 12
- e\_destroy, 12

- e\_quit, 108
- Edit, 77, 78, 88
- Edit menu, 19, 101
- editing
  - code, 4, 14
  - connections, 57
  - creation order, 74
  - menubars, 20
  - userdata, 86
  - userdata labels, 87
- Enable Clear, 70
- Enable Menu, 70
- environments
  - development, 52
- Even Horizontal Spacing, 63, 105
- Even Vertical Spacing, 63, 105
- event handler member functions, 10, 12, 47
- event tags, 14, 57, 106
- events
  - update, 45
- EVNT prefix, 57, 106
- examples directory, 52
- Exit, 100
- external files, 97

## F

- features, of XVT-Design++, 4, 53
- File menu, 19, 99
- files
  - binary, 53
  - C++ source, 93
  - configuration, 111, 116
  - external, 97
  - generating, 91, 94
  - header, 94
  - help text, 94
  - names in XVT Design++, 92
  - overwriting, 95
  - project, 18, 52, 53
  - resource, 94
  - source, 2, 97
  - types of generated, 93
- Focus tag, 107

Font menu, 19, 105

Font tag, 107

## **G**

Generate Application, 52, 100

generated code, 14

Generated Files, 100

generating

code with userdata, 89

files, 91

source code, 50

geometry

attributes, 13

Grid, 105

Grid command, 63

Grid Spacing, 64

gridDisplay, 113

gridNative, 113

grids, 63

character-based, 64

default settings, 113

superimposing, 64

gridSnap, 113

gridX, gridY, 113

GUI

applications, 2

container classes, 12

objects, 12, 13

## **H**

header files, 94

height, of objects, 66

Hello application, 18

testing, 40

help

customer support, 5

on-line, 3

text file, 94

Help tag, 108

hierarchical menus, 75

hierarchy

class, 8

Horizontal Scrollbar, 73

HScroll tag, 107

## **I**

Iconizable, 73

Iconized, 73

if statements, 11

#include, 50, 108

inheritance, multiple, 8

Init, 12, 108

initialAboutBox, 116

installation, 2

instantiating

subclasses, 11

integrated code editing, 4

interface objects, 13, 106, 109

adjusting, 42

attributes, 13

identifier string, 13

setting attributes, 65

size & position, 13

title string, 13

Invisible, 67, 71, 73

Is Quit OK tag, 108

## **J**

Justification, 68

## **K**

Keys, 81

## **L**

Layout menu, 61, 104

layout windows, 60

ListResStrings, 85

## **M**

macCreatorCode, 115

macFileType, 115

Macintosh, 112, 115

macros, 114

Main Code tag, 108

make, 52

Make Same Size, 63, 105

makefiles, 52, 94, 95

- generating, 4
- template macros, 114
- manual
  - conventions used in, 3
  - organization of, 2
  - using, 2
- Margin, 70
- Maximized, 73
- member functions
  - defining, 43
  - event handler, 10, 47
  - for a new class, 71
- memory allocation, 44
- Menu, 102
- Menu Editor, 20, 75, 77
- Menubar, 75
- Menubar Editor, 20, 76, 102
  - invoking, 76
- menubars
  - associating with windows, 28
  - creating, 19
  - initializing, 44
- MenuItem subclasses, 111
- menus, 75
  - adding items to, 22
  - changing title, 21
  - commands, 99
  - Control, 103
  - creating, 21
  - Edit, 19, 101
  - File, 19, 99
  - Font, 19, 105
  - hierarchical, 75
  - Layout, 61, 104
  - setting attributes, 80
  - standard, 19
  - Style, 19, 105
  - TestMode, 90
  - Tool, 102
  - Window, 103
- message window, 37
- Mnemonic, 81

- mnemonics, 22
- Modal/Modeless, 71
- modules, in ACE, 15, 55
- Motif platforms, 73, 86
- Mouse\_Dbl tag, 107
- Mouse\_Down tag, 107
- Mouse\_Move tag, 107
- Mouse\_Up tag, 107
- MS-Windows, 106, 112
- multiple inheritance, 8
- Multiple Selections, 68

## N

- native controls, 5
- New, 77, 79
- New Dialog, 103
- New Project, 99
- New Window, 103
- No Connection, 59

## O

- object-oriented programming, 42
- objects
  - behavior, 11
  - creating, 12
  - delegate, 11
  - GUI, 12
  - height, 66
  - in ACE, 15, 56
  - interface, 13, 65, 106, 109
  - text edit, 68
  - user-defined, 58
  - width, 66
- One Paragraph, 69
- on-line help, 3
- Open Project, 99
- OS/2, 112
- Overtime, 69

## P

- Paste, 49, 57, 70, 78, 101
- PM platforms, 73
- pointers, 43

- portability, 53
- pre-defined dialogs, 38
- proj1.prj, 26
- Project Attributes, 101
- project files, 52, 53
- projectFileExtension, 115
- projects

- closing, 100
  - creating, 18, 54
  - multiple, 54
  - new, 99
  - opening, 99
  - saving, 26, 100
  - testing, 89

- pull-down menus, 75

- push buttons, 28

## **Q**

- Quit, 100

- Quit tag, 107

- QuitOK, 108

## **R**

- radio buttons, 30

- Read-only, 67, 70

- recursion, 12

- Rename, 76

- Replace, 88

- resource identifier attribute, 13

- resources

- #define string, 13
  - creating, 60
  - definition & layout, 4
  - files, 94
  - string, 82

- Revert, 60

## **S**

- Save Project, 100

- Save Project As, 100

- saving projects, 26

- scrollbars, 73

- Select All, 101

- Separator, 79

- Size Box, 73

- Size tag, 107

- Snap To Grid, 64

- source code, 42

- source files, 2

- spacing, controls, 63

- SPCL prefix, 57, 107

- special tags, 14, 57, 106, 107

- staggering windows, 44

- Standard Size, 66

- Std Menus, 79

- string lists, 83

- creating, 84

- editing elements, 84

- string variables, 43

- strings, 82

- creating, 82

- userdata, 85

- Strings Editor, 102

- Style menu, 19, 105

- subclasses, 8, 9, 11

- submenus, 24, 75

- superclasses, 8, 111

- switch statements, 11

## **T**

- tabStop, 116

- tags, 14, 106

- event, 57, 106

- in ACE, 15

- paired with interface objects, 109

- special, 14, 57, 106, 107

- Task Menubar, 33, 35

- Task Window, 13

- Task Window Title, 33

- TASK\_MENUBAR, 20

- templates, 95, 114

- custom names, 114

- customizing, 96

- testing projects, 89

- TestMode, 4, 15, 40, 89

- entering, 89

- leaving, 90
  - special considerations, 91
- text
  - defaults in ACE, 115
- text edit objects, 68
- text-editing pane, 35, 57, 115
- Timer tag, 107
- Title, 80
- title attribute, 13, 65
- titles
  - radio buttons, 31
- Tools menu, 102
- Top tag, 108
- tutorial, for Design++, 17
- .txt files, 108

**U**

- UNIX, 112
- Up/Down, 79
- update events, 45
- Update tag, 107
- URL files, 65, 85, 87, 89, 94, 108
- User Header tag, 108
- user interface
  - code, 14
  - objects, 9
- User tag, 107
- User URL tag, 108
- Userdata, 78, 102
- userdata, 85
  - creating, 86
  - editing, 86
  - labels, 87
  - statements, 89
- Userdata Labels, 101
- user-defined objects, 58
- userMacros, 114
- userTemplates, 114

**V**

- variables
  - defining, 42
  - in action code, 111

- initializing, 42
  - string, 43
- Vertical Scrollbar, 73
- VScroll tag, 107

**W**

- width, of objects, 66
- Win platforms, 73
- WIN\_101, 28
- Window menu, 103
- windows, 27
  - attributes, 27, 72
  - creating, 27, 60
  - layout, 60
  - message, 37
  - multiple similar, 11
  - size & position, 13
  - staggering, 44
- Word Wrap, 69

**X**

- X, Y coordinate locations, 65
- xdContainer data member, 111
- xdReturnValue, 111
- XVT++
  - class hierarchy, 8
  - overview, 7
  - typical application in, 9
- XVT++ Class Library Reference, 7, 43
- XVT++ hierarchy, 111
  - extending, 9
- XVT\_ChildBase::SetEnabledState, 73
- XVT\_ChildBase::SetVisibleState, 73
- XVT\_Control::SetEnabledState, 67
- XVT\_Control::SetVisibleState, 67
- XVT\_Dialog::SetEnabledState, 71
- XVT\_Dialog::SetVisibleState, 71
- XVT\_GlobalAPI::GetResString, 83
- XVT\_GlobalAPI::ListResStrings, 85
- XVT\_MenuItem::e\_action, 75
- XVT\_MenuItem::SetCheckedState, 81
- XVT\_MenuNode::SetEnabledState, 81
- XVT\_StrList, 85

## *Using XVT-Design++*

XVT\_TaskWin, 108

XVTAppWin101, 44

XVT-Design++

- and XVT++ Class Library, 7

- building applications, 17

- features, 4

- installing, 2

- on-line help, 3

- overview, 1

XVTDIR macro, 97

# Platform-Specific Books





# **XVT CUSTOMER SUPPORT**

---

**XVT - THE PORTABLE GUI DEVELOPMENT SOLUTION**

---



## Copyrights

© 1992 - 1993 XVT Software Inc. All rights reserved.

The XVT application program interface, XVT manuals and technical literature may not be reproduced in any form or by any means except by permission in writing from XVT Software Inc.

XVT is a trademark of XVT Software Inc. Other product names mentioned in this document are trademarks or registered trademarks of their respective holders.

## Published By

XVT Software Inc.  
Box 18750  
Boulder, CO 80308  
(303) 443-4223  
(303) 443-0969 (FAX)

## Printing History

First Printing.....	March, 1989 .....	XVT Version 1.2
Second Printing .....	January, 1990 .....	XVT Version 2.0
Third Printing .....	May, 1992 .....	XVT Version 3.0
Fourth Printing .....	May, 1993 .....	XVT Version 3.0

This manual is printed on recycled paper by  
Continental Graphics, Broomfield, Colorado.



# 1

---

## Customer Support

If you have problems or questions while using XVT products, you can talk to an XVT Customer Support Engineer. Customer Support is available to customers who have purchased an XVT product and have a current maintenance contract.

Please note that only one individual per purchased copy of an XVT product may request support. Questions will be taken only from the individual named on the software registration form.

Feel free to contact XVT Customer Support if you have a question, or would like to suggest a software enhancement or a change to any document. Your call is always welcome.

### European Customers

If you are a European customer, Customer Support is available through your distributor. If you purchased your XVT product through a European distributor, please contact them for support.

## 1.1. How to Contact Customer Support

You can contact XVT Software Customer Support in several different ways:

- Telephone us at (303) 443-8988 (8 AM to 5 PM, MST, Monday-Friday)
- Send us electronic mail via the Internet at [techsup@xvt.com](mailto:techsup@xvt.com)
- Send us electronic mail via CompuServe Mail at 75765,1233
- Send us electronic mail via our Bulletin Board System (BBS) at (303) 443-9083 (2400 baud) or (303) 443-7780 (9600 baud).

- Write us at XVT Software Inc., P.O. Box 18750, Boulder, CO 80308

**Note:** Before you can use the XVT Bulletin Board System, you must contact XVT Customer Support to get a BBS login.

## 1.2. Information We Need to Help You

When you contact XVT Customer Support, please supply the following information:

- The name and version number of the product (for example, XVT/Win 3.01)
- The serial number of the product (found on your distribution media)
- Your platform type (for example, Sun-4, IBM RS/6000)
- The operating system and version number (for example, HP-UX 9.0, SunOS 4.1.3)
- The compiler and version number (for example, THINK C 3.0)
- The window manager and version number (for example, MS-Windows 3.1)
- A detailed description of the problem, including the number displayed with any internal error message such as this:  
Internal XVT Error: 37007-408323

**See Also:** For more information about internal error messages, see section 1.5.3, "Error Messages File."

## 1.3. What XVT Customer Support Provides

XVT Customer Support can serve you better if you understand what services are available.

This is what XVT's Customer Support can do:

- Provide "tips" to help you effectively use XVT functionality
- Explain XVT functionality and specifications
- Diagnose and analyze XVT-related application problems
- Suggest workarounds
- Suggest how to access native window system development tools
- Collect feedback for future product development

Keep in mind that XVT Customer Support cannot do the following:

- Design customer applications
- Debug user code
- Explain how to use operating systems, window systems, or compilers (except with regard to XVT application resources)
- Explain how to use native window system development tools
- Extensively teach XVT programming

**See Also:** If you need more help than Customer Service can provide, consider contacting XVT's Consulting and Training Group. See "Consulting and Training Services," at the end of this chapter.

## 1.4. How Customer Support Works

XVT's Customer Support goal is to respond to all requests within twenty-four hours. As soon as we log your call into our system, you will receive a service request number.

If we have questions about your request, we ask that you respond to them within five working days. Please let us know if you need more time; otherwise, if we receive no response from you after five days, your service request will be closed.

## 1.5. Troubleshooting

This section describes some resources that you can use to diagnose problems or answer questions about XVT products. We suggest that you try to obtain help from one of these sources before you call Customer Support.

### 1.5.1. XVT Example Set

With each release XVT provides an Example Set, which contains application programs that demonstrate the functionality of the XVT library. Using the Example Set, you can

- Verify that your XVT installation is correct by building and running all the examples in the **examples** directory (includes makefiles or project files where applicable)
- See a working model when you are learning XVT
- Compare sample functions to functions in your code, to verify correct behavior

XVT Customer Support may ask you to use the Example Set to verify a problem when you report it.

### 1.5.2. XVT Documentation

To help you use XVT products, XVT provides several kinds of documentation:

- Technical manuals (supplied in binders such as this one)
- **readme** files (supplied with your distribution media)
- Technical Notes (published periodically)
- Errata (lists of corrections published periodically)
- Change Pages (corrected pages that you can insert into your manuals)

The documentation is designed to give you the information you need to use XVT products at all levels, from introductory (the *Tutorial and Guide*) to advanced (the *Programmer's Reference*). Platform-specific books (PSBs) in the "Install and Usage" section of your binder contain information about your particular platform(s).

XVT takes pride in its documentation, and continually seeks to improve it. If you find a documentation error, please contact Customer Support. They will forward your suggestion to XVT's documentation team.

#### 1.5.2.1. Suggesting a Technical Note

Included with every Technical Notes packet is a form for suggesting a new Technical Note. You can also download this form from the XVT BBS. We welcome your suggestions.

### 1.5.3. Error Messages File

The XVT library does some internal error checking. If you use an XVT function incorrectly, you may see an "Internal XVT Error" dialog box. This dialog box displays the error number followed by the timestamp of the product release version (for example, "Internal XVT Error: 37007-408323").

To find a description of the error, look in the file named **errcodes.txt** (**ERRCODES.TXT** on some platforms), in the **doc** directory of your release. The description might give you enough information to resolve the problem. If you contact XVT Customer Support about the problem, please mention the error number and the description of the error.

## **1.6. Consulting and Training Services**

The XVT Consulting & Training Group (CTG) offers extensive fee-based services to help customers use XVT products. CTG specialists can help you learn GUI programming, or help you prototype, design, code, debug, and maintain your XVT applications.

In addition to consulting, CTG specialists also conduct on-site and public training classes in XVT and GUI programming techniques.

**See Also:** For more information about the XVT Consulting & Training Group, call (303) 443-4223.

